AMSTRAD

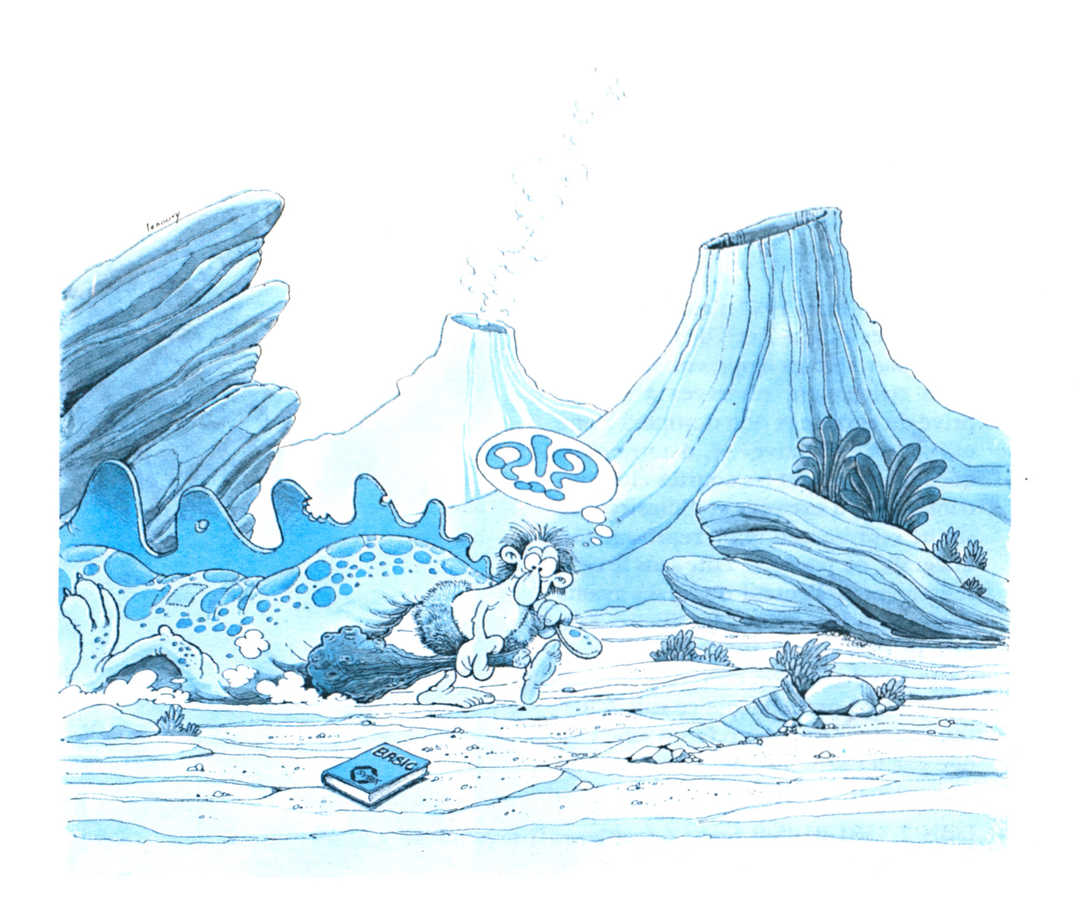
PREMIERS PROGRAMMES

RODNAY ZAKS



AMSTRAD

PREMIERS PROGRAMMES



Illustrations couverture et intérieur : **Daniel Le Noury**
Traduction **Annie Laurent**

ISBN 2-7361-0105-9



RODNAY ZAKS



Paris • Berkeley • Düsseldorf • Londres

SOMMAIRE

1

Parler BASIC

2

Communiquer avec votre ordinateur

3

Les calculs en BASIC

4

Mémoriser des valeurs et utiliser des variables

5

Écrire un programme clair

6

Prendre des décisions

7

Automatisation des répétitions

8

Créer un programme

9

Étude de cas : conversion métrique

10

L'étape suivante

Annexes

PREFACE

Des centaines, voire des milliers de livres, ont été écrits sur le BASIC.

Pourquoi un de plus ? Tout simplement parce que ce nouvel ouvrage est destiné à un public nouveau.

Jusqu'ici, en effet, seuls les quelques techniciens qui avaient accès aux ordinateurs utilisaient les langages de programmation tels que le BASIC. Ces programmeurs constituaient ainsi une élite.

Avec l'apparition puis la vulgarisation de l'ordinateur personnel, le BASIC est devenu le langage informatique le plus largement utilisé et le plus accessible. C'est ce langage que doivent parler tous ceux - de plus en plus nombreux - qui, désormais, se servent couramment de l'ordinateur à des fins ludiques, éducatives, commerciales ou professionnelles.

Ce livre, original dans sa conception et dans sa présentation, s'adresse à ces nouveaux utilisateurs, âgés de 7 à 77 ans, et qui, dépourvus de toute connaissance technique, désirent apprendre rapidement à se servir du BASIC.

L'auteur ne leur demande rien au départ qu'un peu de vigilance et l'enthousiasme du néophyte. Il a recherché, à leur intention, une approche simple, directe ou méthodique, avec l'espoir de faciliter leur apprentissage. Son objectif est de leur enseigner l'essentiel en quelques heures ; l'essentiel, c'est-à-dire : écrire un premier programme BASIC et, bientôt, réaliser aisément des programmes personnels cohérents et efficaces.

RODNAY ZAKS
Berkeley, janvier 1983

COMMENT LIRE CE LIVRE

C'est un manuel pratique. Lisez les chapitres dans l'ordre où ils ont été écrits et n'abandonnez un chapitre pour le suivant que lorsque vous serez sûr de l'avoir compris. A la fin de chaque chapitre, vous trouverez des exercices d'application qui vous permettront de vérifier vos acquisitions. Nous vous conseillons d'en faire le plus possible. L'Annexe A à la fin du livre contient les corrigés de ces exercices.

Essayez tous les programmes. Rien ne vaut la pratique, l'expérimentation, pour bien fixer les connaissances. Ce livre vous enseignera tout ce qu'un débutant doit savoir, mais n'oubliez pas que rien ne peut remplacer l'expérience.

Pour atteindre l'objectif que l'on s'est fixé en écrivant ce livre, on a été amené, afin de vous faciliter la tâche, à faire des choix. Ainsi le livre ne prend pas en compte toutes les caractéristiques et tous les concepts du BASIC de l'Amstrad, mais uniquement ceux qui ont été jugés les plus importants.

Nous souhaitons que cet ouvrage vous soit une aide efficace et vous permette de réaliser rapidement et sans difficulté vos propres programmes BASIC.

CE QUE VOUS ALLEZ APPRENDRE

LE CHAPITRE 1 traite du langage des ordinateurs et vous présente les héros de ce livre : l'Ordinateur, l'Interpréteur, le Programme, les Instructions et autres importants personnages.

LE CHAPITRE 2 indique la manière de communiquer avec l'ordinateur au moyen du clavier et de l'écran. Vous apprendrez à écrire vos premiers programmes BASIC et à les exécuter.

LE CHAPITRE 3 vous explique comment faire des calculs avec le BASIC.

LE CHAPITRE 4 vous aidera à écrire des programmes pouvant être indéfiniment réutilisés. En outre, vous y apprendrez à vous servir des variables correctement et efficacement.

LE CHAPITRE 5 vous montrera comment réaliser des programmes clairs et lisibles.

LE CHAPITRE 6 vous apprendra à prendre des décisions en fonction de certaines valeurs numériques ou logiques.

LE CHAPITRE 7 est une initiation à l'automatisation des tâches répétitives, au moyen des boucles.

LE CHAPITRE 8 suit les principales étapes de la conception d'un programme : mise au point de l'algorithme, construction de l'organigramme, écriture et documentation du programme.

LE CHAPITRE 9 vous aide à mettre en application tous ces concepts dans une étude de cas concrète.

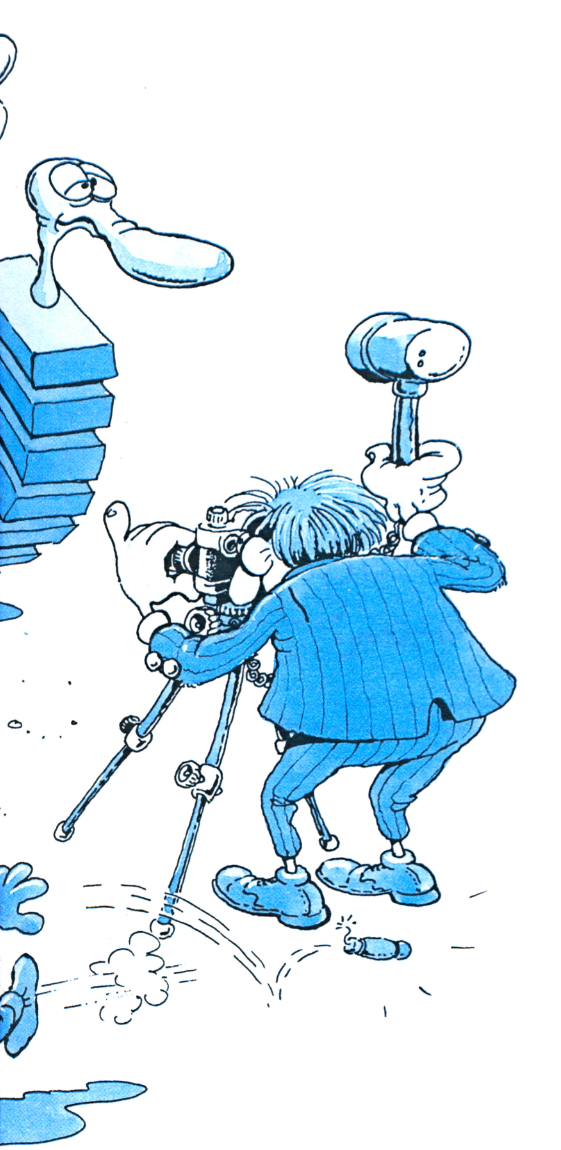
LE CHAPITRE 10 donne un aperçu de certaines techniques de programmation plus complexes.

LES ANNEXES A, B et C contiennent les corrigés des exercices, une liste des «mots réservés» du BASIC de l'Amstrad et un glossaire.

Si vous êtes prêt, ouvrons l'album de famille. Voici donc nos personnages...

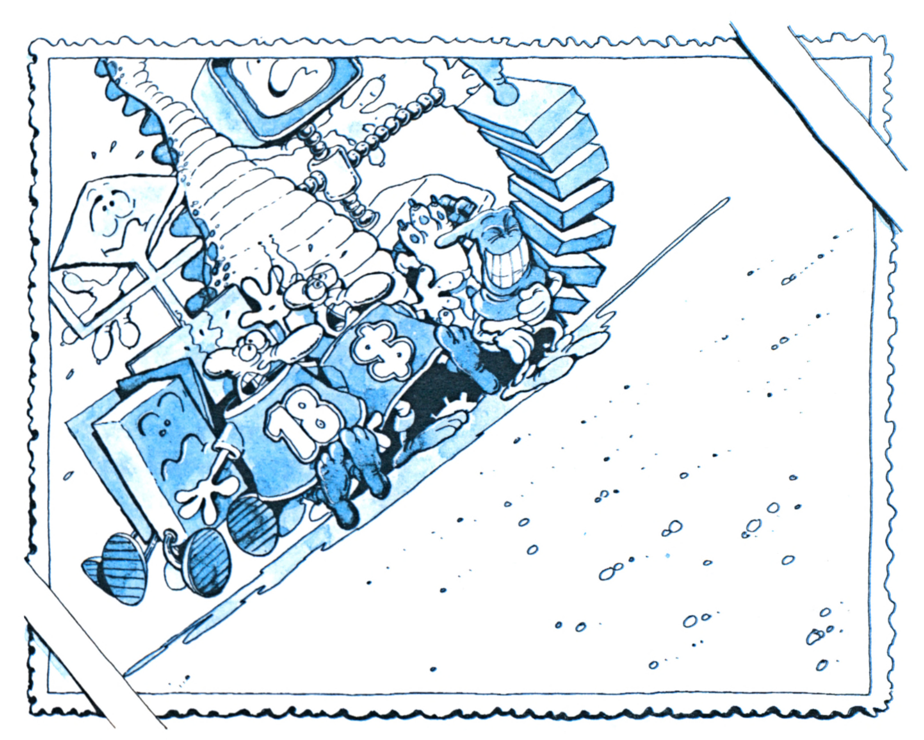

18
$

FAITES CONNAISSANCE AVEC NOS HEROS



Dino le Programmeur, prenant appui sur l'indispensable Organigramme est entouré de : l'Interpréteur BASIC juché sur son ami l'Ordinateur, le Serpent Programme, un Bug diabolique, deux Variables, et quelques Instructions prêtes à rejoindre leur poste.

Hé ! Où court-il celui-là ? Avec les Bugs on peut s'attendre à tout !

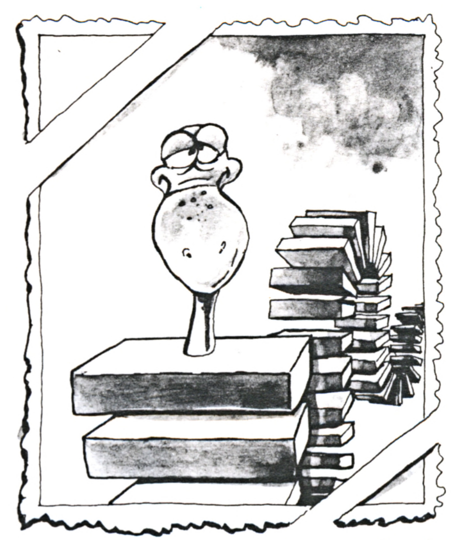


Et voilà ! Encore un sale coup du Bug. En fait de photo de groupe !

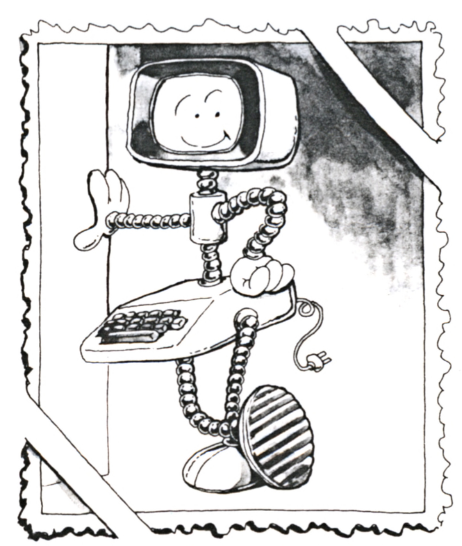
L’ALBUM DE FAMILLE



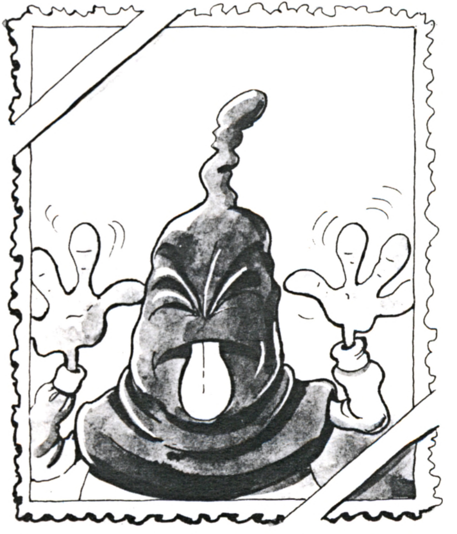
Voici l'Interpréteur BASIC. Quand il ne dort pas, il occupe la mémoire de votre Ordinateur. Sa tâche consiste à traduire vos instructions à l'ordinateur. Il fera de son mieux pour vous être utile.



Non, ce n'est pas un monstre - c'est le Serpent Programme. Il est composé d'instructions. Vous apprendrez à les assembler. Avant de s'en faire un ami, il faut d'abord l'apprivoiser. Et surtout, préservez-le des horribles Bugs.



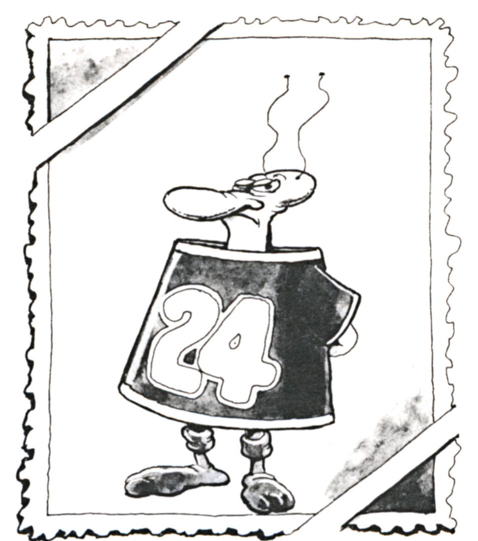
Votre ami l'Ordinateur. A votre service.



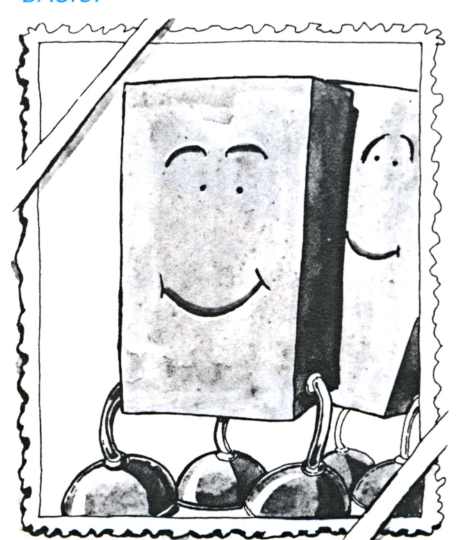
Regardez bien. Ceci est un Bug. Il cherche à vous rendre la vie impossible. Chassez-le impitoyablement de vos programmes.



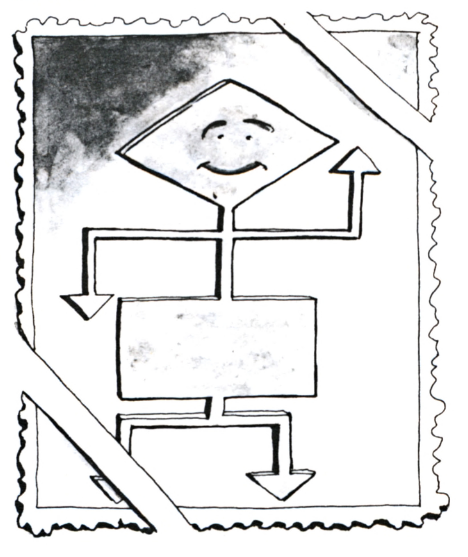
Voici le sympathique Dino. Il n'a pas fait de très longues études et pourtant il saura vous convaincre : ce n'est pas terriblement difficile d'écrire des programmes BASIC.



Ça, c'est une Variable Numérique. On lui a épinglé sa valeur sur le ventre. Si elle n'a pas l'air satisfait, c'est qu'elle voudrait retourner à la case mémoire qui lui est réservée.



Les instructions BASIC, toutes prêtes à rejoindre le Serpent Programme.



Voici l'Organigramme, votre meilleur ami. Il vous aidera à réaliser des programmes qui fonctionnent.



1

PARLER BASIC

Nous avons déclaré dans la préface qu'en une heure vous pouvez apprendre à écrire un programme BASIC. Et pourtant nous commencerons par un chapitre traitant des concepts et des définitions. Ce ne sera pas du temps perdu. Il s'agit d'*acquérir,* et de *retenir* des connaissances, et donc, tout d'abord, de *comprendre.* Ce chapitre vous aidera à mieux comprendre ce qu'est la programmation, la réalisation d'un programme BASIC, et à vous familiariser avec le vocabulaire des ordinateurs.

Nous allons donc :
— poser certaines notions et définitions fondamentales ;
— apprendre à donner des ordres à un ordinateur, c'est-à-dire *programmer* ;
— voir pour quelles raisons nous sommes contraints de recourir à des «langages de programmation» tels que le BASIC ;
— déterminer la fonction de l'*interpréteur* ;
— faire un très court historique du BASIC et de ses dialectes et indiquer sommairement ses utilisations possibles ;
— et enfin décrire les composants de tout *système informatique.*

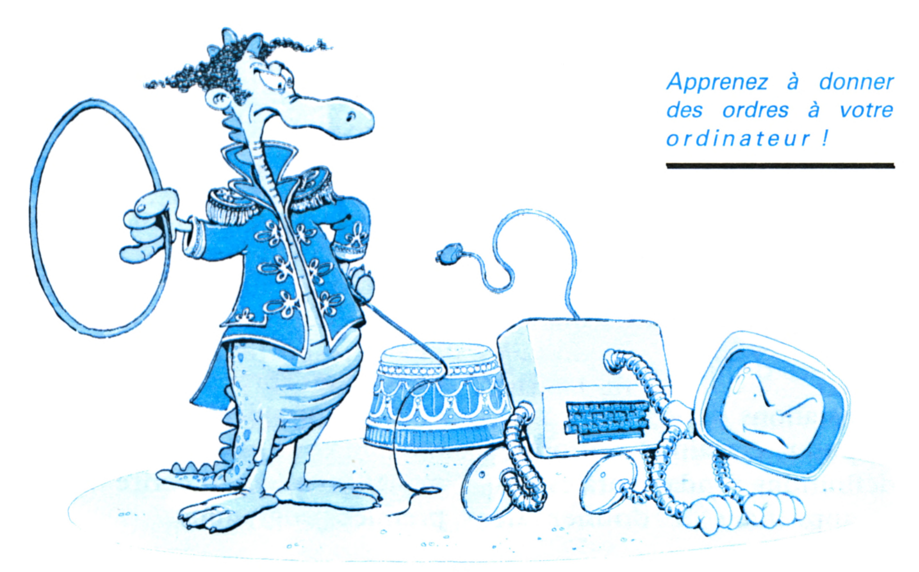
Ce n'est qu'après avoir bien assimilé ces premières connaissances que vous pourrez véritablement envisager d'écrire votre premier programme.

La programmation

Votre ordinateur est une machine conçue pour traiter de l'*information* qui peut être du texte ou des nombres. Vous pouvez aussi bien lui demander d'afficher des mots et des phrases sur un écran que de faire des calculs plus ou moins complexes ; dans le premier cas il s'agit de *traitement de texte,* dans le second, de *traitement numérique.* Pour que l'ordinateur puisse effectuer ce travail, il faut lui donner des instructions dans le «langage» qu'il comprend. Chaque ordinateur ne peut «comprendre» (c'est-à-dire reconnaître et exécuter) qu'un nombre relativement réduit d'instructions différentes (quelques centaines).

Les instructions qu'un ordinateur peut comprendre directement sont en *langage machine.* Elles sont stockées (ou enregistrées) en format *binaire,* c'est-à-dire par groupes de 0 et de 1, dans la mémoire de l'ordinateur. Chaque 0 ou 1 est appelé un *bit,* et chaque groupe de huit bits un *octet.*

Un *programme* est une suite d'instructions qui a pour objectif la réalisation d'une tâche déterminée. (Si la suite des instructions données ne conduit pas à la réalisation de la tâche proposée, c'est qu'il y a une *erreur*.)



Apprenez à donner des ordres à votre ordinateur !

L'ordinateur n'exécute pas immédiatement et globalement toutes les instructions qu'il a reçues, mais successivement et dans l'ordre où elles lui ont été données. L'écriture d'un programme (la suite des instructions) en langage machine (c'est-à-dire en format binaire) est un travail long et fastidieux.

Bien sûr, on aimerait pouvoir donner ces instructions à l'aide du langage habituel, écrit ou parlé. C'est un langage que l'ordinateur ne comprend malheureusement pas. Il exige en effet que les instructions soient claires et sans ambiguïté, présentées dans un ordre déterminé, avec une logique et une précision rigoureuses. Or, chacun sait que notre langage ordinaire est, tout au contraire, imprécis et ambigu, peu soucieux d'ordonnance logique et que le sens, pour une bonne part, est commandé par le contexte écrit ou par la situation et le comportement du locuteur.

N'espérez pas pouvoir un jour demander à votre robot-ordinateur d'aller à la cuisine vous préparer un œuf à la coque si vous ne lui avez pas préalablement indiqué dans les moindres détails la succession des opérations à effectuer. Et si, en application d'une programmation compliquée, difficile, il réussit à vous préparer votre œuf à la coque, il sera incapable de réaliser le même exploit dans la cuisine de votre voisin, où tout est disposé différemment. Les ordres donnés aux ordinateurs doivent être d'une extrême précision.

Afin de pouvoir donner de tels ordres, on a mis au point des *langages simplifiés.* Le langage binaire (appelé aussi langage machine) est celui que l'ordinateur comprend le mieux. Ce langage binaire, qui convient si bien à l'ordinateur, n'est malheureusement pas pour l'homme d'un emploi très aisé. Il a donc fallu inventer d'autres langages qui satisfassent tout à la fois l'homme et la machine, et facilitent entre eux la communication. Ces langages, qui se rapprochent de l'anglais courant, sont appelés *langages évolués.*

On n'y utilise qu'un nombre limité de mots anglais. Chacun de ces mots correspond à une commande prédéfinie. Les phrases, ou plutôt les *instructions* données, doivent respecter une grammaire stricte : la *syntaxe* du langage.

Tout *langage de programmation* résulte de la combinaison d'un vocabulaire restreint et d'une syntaxe.

Le BASIC est un *langage de programmation,* c'est-à-dire un ensemble de règles (la syntaxe), de mots et de symboles (le voca-



«J'adore le BASIC ! S'il vous plaît, parlez-moi BASIC.»

bulaire) qui permet d'adresser des instructions à un ordinateur. Plusieurs instructions successives constituent un *programme.*

Supposez que nous voulions additionner 2 + 2 et afficher le résultat.

En BASIC, nous écririons :

```
1 R = 2 + 2
2 PRINT R
```

R voulant dire «résultat».

Mais... nous avons dit plus haut que le seul langage qu'un ordinateur peut comprendre directement est le *langage machine* et nous envisageons maintenant de donner des ordres à un ordinateur dans un langage proche de l'anglais.

La contradiction n'est qu'apparente. En effet, l'ordinateur à lui seul ne peut pas comprendre le BASIC directement, ni d'ailleurs aucun *langage de programmation évolué* (un langage qui emploie des phrases semblables à l'anglais). Donc, pour être compris par un ordinateur, un programme écrit dans un langage évolué tel que le BASIC

«Vous me reconnaissez ? Je suis le Serpent Programme et je suis composé d'instructions.»

doit être *interprété* par un programme spécial : l'*interpréteur*. En d'autres termes, vous parlerez BASIC avec votre ordinateur par l'intermédiaire d'un interpréteur. Donc, pour exécuter un programme en BASIC votre ordinateur doit contenir un interpréteur BASIC. Voyons quel est son rôle.

L'interpréteur BASIC

L'interpréteur BASIC lit chaque instruction BASIC que vous tapez au clavier, l'analyse et l'exécute selon ses propres procédures.

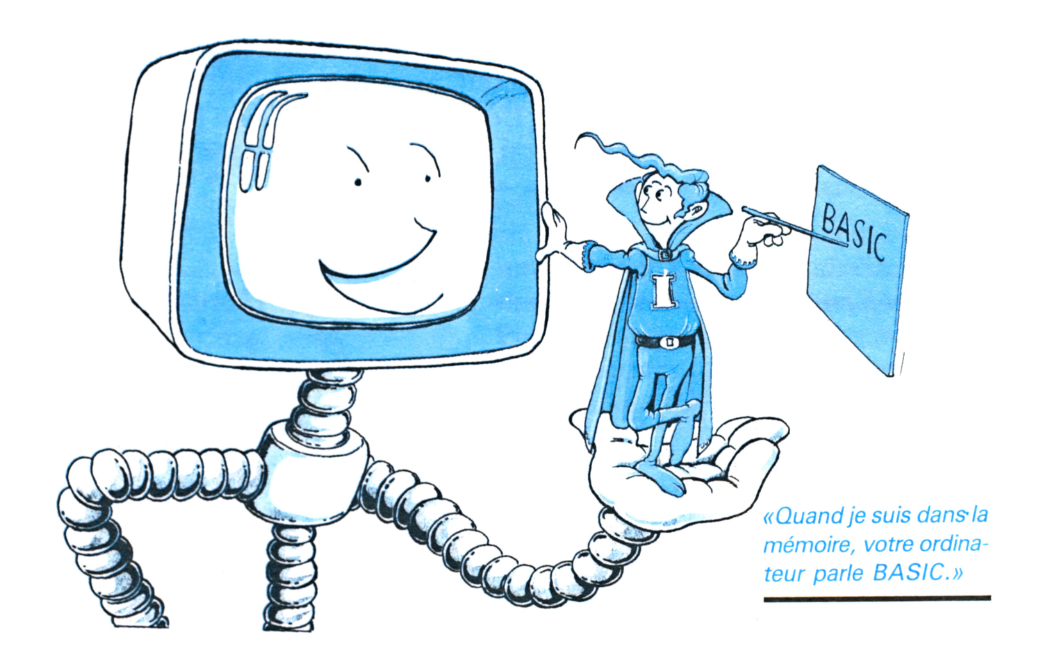
Cette opération a lieu à votre insu (c'est-à-dire qu'elle a lieu à l'intérieur de votre ordinateur). Quand vous activez le programme interpréteur, votre ordinateur se comporte comme s'il parlait BASIC. Il peut aussi parler d'autres langages de programmation (si on lui fournit les interpréteurs appropriés).

Il y a plusieurs sortes d'interpréteurs BASIC utilisables sur votre ordinateur. On les classe en deux catégories : *résidents* et *non résidents*.

L'Amstrad comme la plupart des petits ordinateurs est pourvu d'un interpréteur BASIC *résident*. Il est dit résident parce qu'il réside en permanence dans la mémoire de l'ordinateur. Il est disponible immédiatement lorsque l'ordinateur est mis sous tension. Quand l'indicatif du BASIC (Ready) apparaît sur votre écran, cela signifie que l'ordinateur est prêt à exécuter vos instructions BASIC.



«Vous me reconnaissez ? Je suis l'interpréteur BASIC. Toujours prêt à traduire vos instructions à l'ordinateur, je suis un programme et je loge dans la mémoire de l'ordinateur.»



«Quand je suis dans la mémoire, votre ordinateur parle BASIC.»

Voyons à présent ce qu'est le BASIC, comment il a été inventé et les dialectes qui en découlent.

Qu'est-ce que le BASIC ?

Les langages évolués ont été créés pour qu'un utilisateur puisse donner facilement des instructions à un ordinateur, c'est-à-dire pour qu'il le «programme» facilement. Au cours des années, on a inventé des centaines de langages de programmation.

Les premiers ordinateurs étaient utilisés à des fins scientifiques et les premiers langages de programmation ont été conçus pour faciliter les calculs numériques, ainsi l'ancêtre des langages, le FORTRAN (*FORmula TRANslator,* traducteur de formules). Toutefois, le FORTRAN présentait de nombreux inconvénients et on a cherché à créer des langages plus performants. Le BASIC est un de ces langages ; le COBOL, l'APL et le PASCAL sont également très employés.

Le BASIC représente un progrès important en ce qu'il est simple et *conversationnel.*

Le BASIC (*Beginners All-purpose Symbolic Instruction Code*) est un langage de programmation «pour débutants» et pour «tous usages». Il a été inventé en 1964 au Dartmouth College par John KEMENY et Thomas KURTZ dont les recherches étaient financées par une subvention de la Fondation nationale pour la science. Leur but était de concevoir un langage facile à utiliser par des néophytes. Le BASIC ayant été conçu pour être *conversationnel* - en fait il était le premier langage de programmation *conversationnel* -, l'utilisateur pouvait dialoguer avec le programme sur un terminal au lieu d'employer des lots de cartes perforées IBM comme c'était le cas avec les anciens langages. A l'origine, le BASIC a fonctionné au Dartmouth College avec le système en temps partagé GE225. Des terminaux étaient disponibles dans toute l'université et un grand nombre d'utilisateurs pouvaient accéder en même temps à l'ordinateur.

Le succès du BASIC a été rapide. General Electric (GE) a immédiatement décidé de l'employer commercialement. KEMENY et KURTZ ont publié le premier livre sur le BASIC en 1967. Enfin, Hewlett Packard (HP) et Digital Equipment Corporation (DEC) ont pris les mesures nécessaires pour fournir le BASIC sur la plupart de leurs ordinateurs.

Le BASIC présente deux avantages importants par rapport à des langages comme le FORTRAN :

1. **Pour l'utilisateur :** le BASIC est le langage le plus facile à apprendre, en particulier pour un débutant.

2. **Pour le fabricant :** le BASIC est le langage le plus facile à mettre en œuvre sur un ordinateur. Comme le langage est simple, l'interpréteur est simple aussi et n'exige qu'une petite capacité mémoire.

Un troisième facteur a contribué à l'énorme succès du BASIC : l'apparition des micro-ordinateurs à bas prix. Quand, à la fin des années 1970, les micro-ordinateurs ont été plus largement commercialisés, le BASIC est devenu le langage de programmation universel pour ces petits appareils.

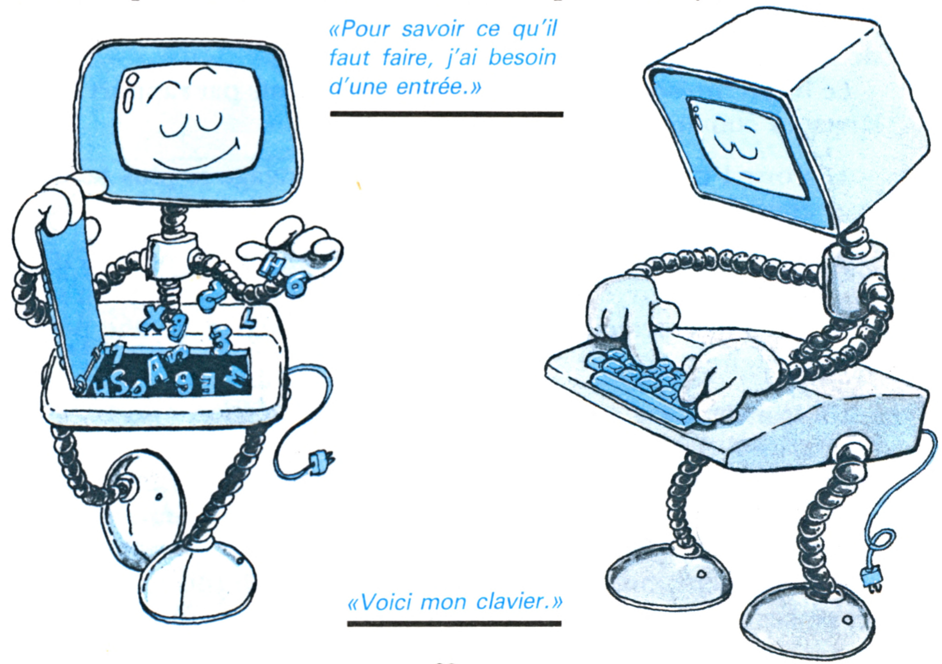
Actuellement, on utilise le BASIC sur presque tous les ordinateurs. Les fabricants ont peu à peu apporté des extensions au lan-

gage et lui ont ajouté des particularités, de sorte qu'aujourd'hui le BASIC est probablement le langage d'ordinateur le moins normalisé. Il n'y a pas deux BASIC semblables. A partir du BASIC original, on a formé une famille de langages. Pour unifier ces divers langages on a souvent proposé des normes, mais aucune des solutions envisagées ne s'est imposée et pour le moment, aucune n'a de chances d'y parvenir. Cela ne veut pas dire que l'on doive réapprendre le BASIC chaque fois que l'on change d'ordinateur. Lorsque l'on connaît le BASIC essentiel (commun à toutes les versions), on peut très rapidement s'initier à chacune des versions particulières, chacune ayant ses caractéristiques propres et ses avantages.

Après cet aperçu sur les langages de programmation en général et sur le BASIC en particulier, parlons de votre ordinateur et de la façon dont il traite l'information.

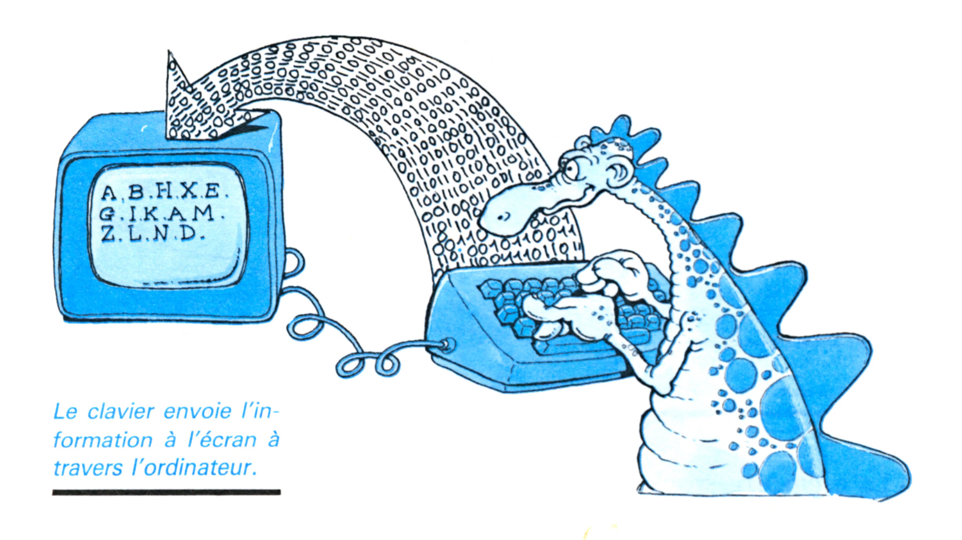
Votre ordinateur

Votre ordinateur traite l'information et communique avec vous par l'intermédiaire d'un clavier et d'un écran et éventuellement d'une imprimante. Le *clavier* est utilisé pour envoyer l'informa-



tion à l'ordinateur : chaque fois qu'une touche est enfoncée, le code électronique correspondant au caractère est envoyé à l'ordinateur qui le reconnaît, l'exécute ou l'ignore. Le clavier est votre *dispositif d'entrée.*

L'*écran* d'un téléviseur ou d'un moniteur affiche l'information générée par le programme. Chaque caractère entré au clavier est d'abord envoyé à l'ordinateur, puis «reproduit» sur l'écran. Il n'y a généralement pas de liaison directe entre le clavier et l'écran. Toutes les informations passent par l'ordinateur.

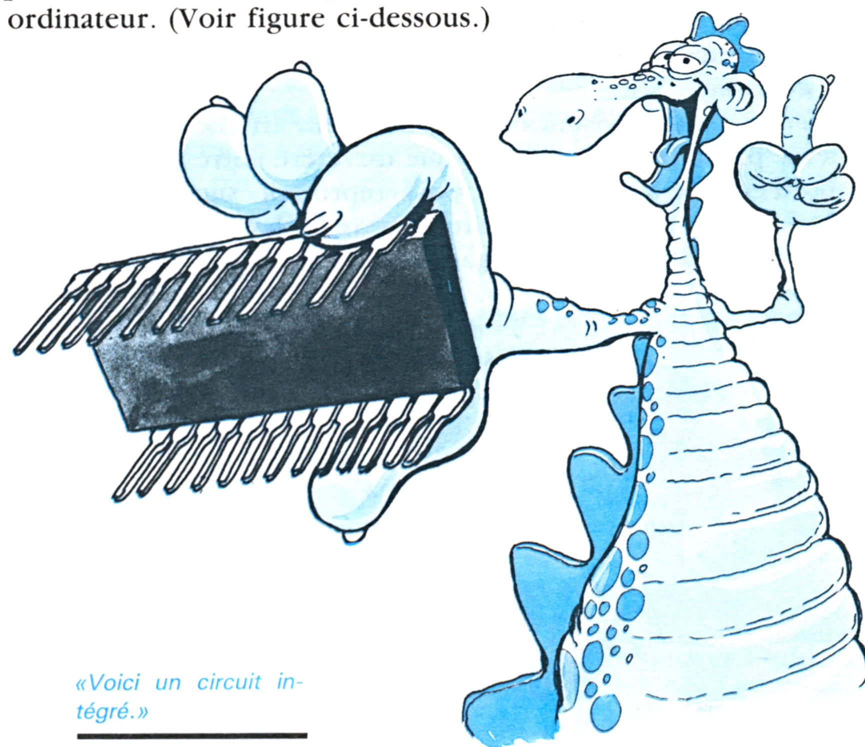


Le clavier envoie l'information à l'écran à travers l'ordinateur.

Dans le cas de l'Amstrad, le clavier et l'ordinateur ainsi que le magnétophone sont intégrés dans un même boîtier. L'écran, l'imprimante et les unités de disques utilisés pour stocker les programmes sont séparés.

L'ordinateur proprement dit comprend une unité de traitement (l'unité centrale), une mémoire, et plusieurs interfaces (les dispositifs électroniques de connexion des imprimantes et autres unités). L'*unité centrale* (CPU, *Central Processing Unit,* Unité Centrale de traitement) recherche dans la mémoire les instructions du programme et les exécute. L'unité centrale est constituée de quelques composants appelés circuits intégrés ou «puces». Le *micro-*

processeur est l'élément principal de l'unité centrale de tout micro-ordinateur. (Voir figure ci-dessous.)



«Voici un circuit intégré.»

La *mémoire* stocke les programmes ainsi que toutes les informations manipulées, lues ou générées par les programmes durant leur exécution. Pour qu'un programme soit exécuté, il faut d'abord le placer dans la mémoire de l'ordinateur. S'il est enregistré sur cassette ou sur disquette, on le transfère dans la mémoire. Cette opération est appelée *chargement* du programme. L'ordinateur doit pouvoir contenir le plus long programme ainsi que toutes les données qu'il va manipuler.

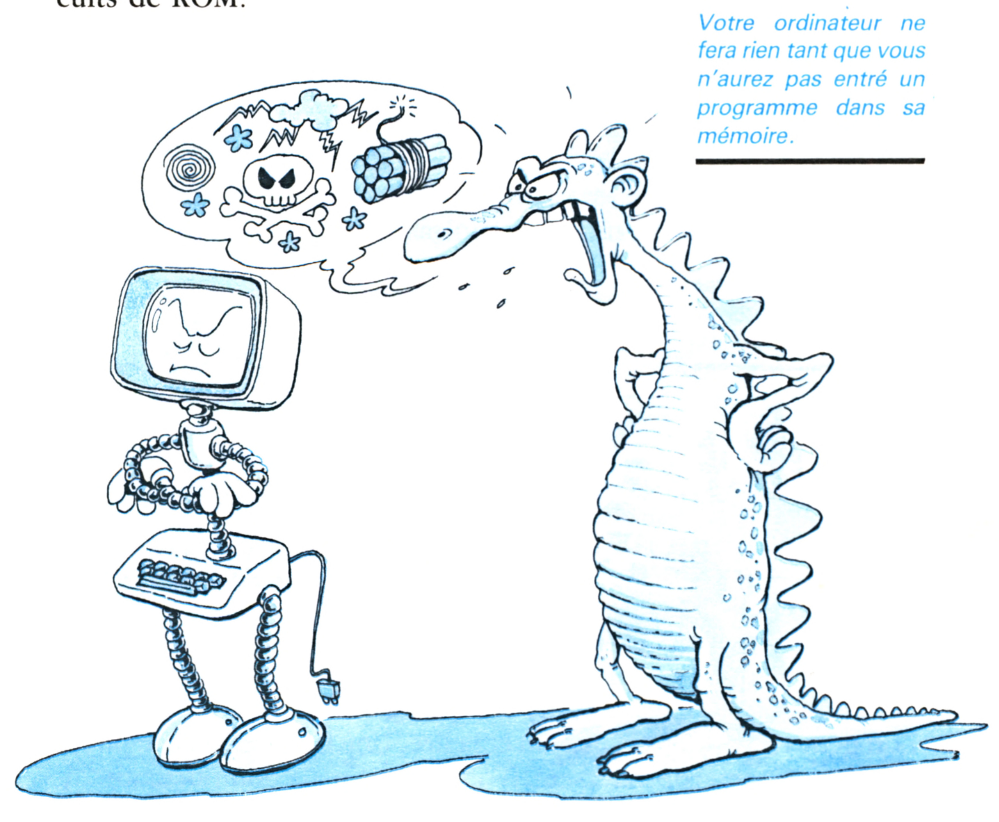
Votre ordinateur comporte deux types de mémoire : la mémoire morte (ou ROM, *Read-Only Memory*) et la mémoire vive (ou RAM, *Random Access Memory*). Le type «normal» de mémoire que vous allez utiliser pour stocker votre programme est la RAM ou mémoire vive. La RAM est une mémoire de lecture-écriture, c'est-à-dire que l'on peut y écrire l'information et la lire. Une RAM ressemble physiquement au microprocesseur (voir figure) mais comporte un cir-

cuit différent. Malheureusement, malgré les progrès techniques, ce type de mémoire est volatile, c'est-à-dire que le contenu de la RAM disparaît lorsque l'on met l'ordinateur hors tension. Lorsque vous avez terminé une opération, et si vous désirez conserver votre programme vous le stockerez sur un support non volatil tel qu'une cassette ou une disquette. Pour exécuter de longs programmes vous aurez éventuellement besoin d'une extension mémoire connectée à votre ordinateur. Le BASIC de l'Amstrad est stocké en ROM.

La ROM est une mémoire morte. Ce type de mémoire contient des programmes figés, enregistrés par le fabricant, et qui ne peuvent être modifiés. C'est une mémoire non volatile et non effaçable.

Vous ne pouvez utiliser la ROM pour y stocker d'autres programmes. Tout programme entré dans l'ordinateur sera chargé dans la RAM. Vous pourrez aussi acheter des cartouches de programmes. Les programmes sont alors stockés dans la cartouche sur les circuits de ROM.

Votre ordinateur ne fera rien tant que vous n'aurez pas entré un programme dans sa mémoire.

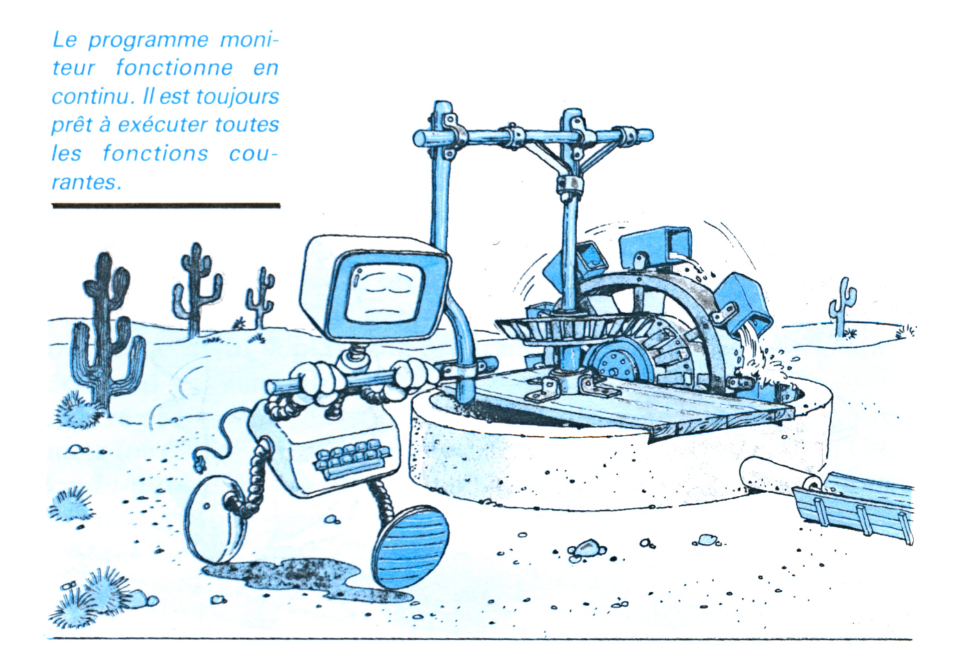


Les deux périphériques dont l'emploi est le plus courant sont la *mémoire de masse* et l'*imprimante.* Ces dispositifs sont reliés à l'ordinateur par l'intermédiaire de systèmes électroniques appelés *interfaces.* L'unité de mémoire de masse peut être une cassette ou une ou plusieurs *unités de disques.* (Ces dispositifs utilisent un support magnétique pour enregistrer les informations et peuvent donc en stocker beaucoup plus que la mémoire électronique interne de l'ordinateur.) Ces dispositifs spéciaux requièrent une interface spécifique dans le boîtier de l'ordinateur leur permettant de communiquer avec lui. L'Amstrad dispose d'une interface intégrée pour une ou plusieurs unités de disques ainsi qu'une imprimante.

Une *imprimante* permet d'avoir une copie sur papier des programmes ou des résultats. C'est donc comme l'écran de visualisation, un *dispositif de sortie.* Le BASIC offre un choix d'instructions bien définies pour envoyer l'information, soit à l'écran de visualisation, soit à l'imprimante.

Nous avons vu le vocabulaire de base. Avant de passer au second chapitre, et de commencer à utiliser l'ordinateur, une mise en garde est nécessaire.

Le programme moniteur fonctionne en continu. Il est toujours prêt à exécuter toutes les fonctions courantes.

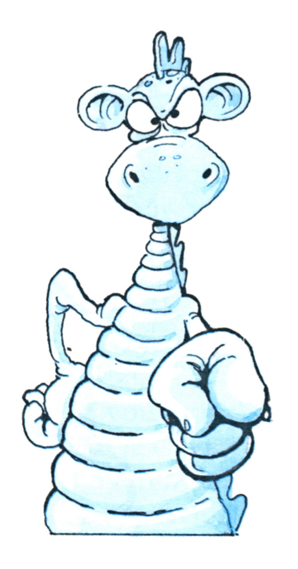


Les ordinateurs et la syntaxe

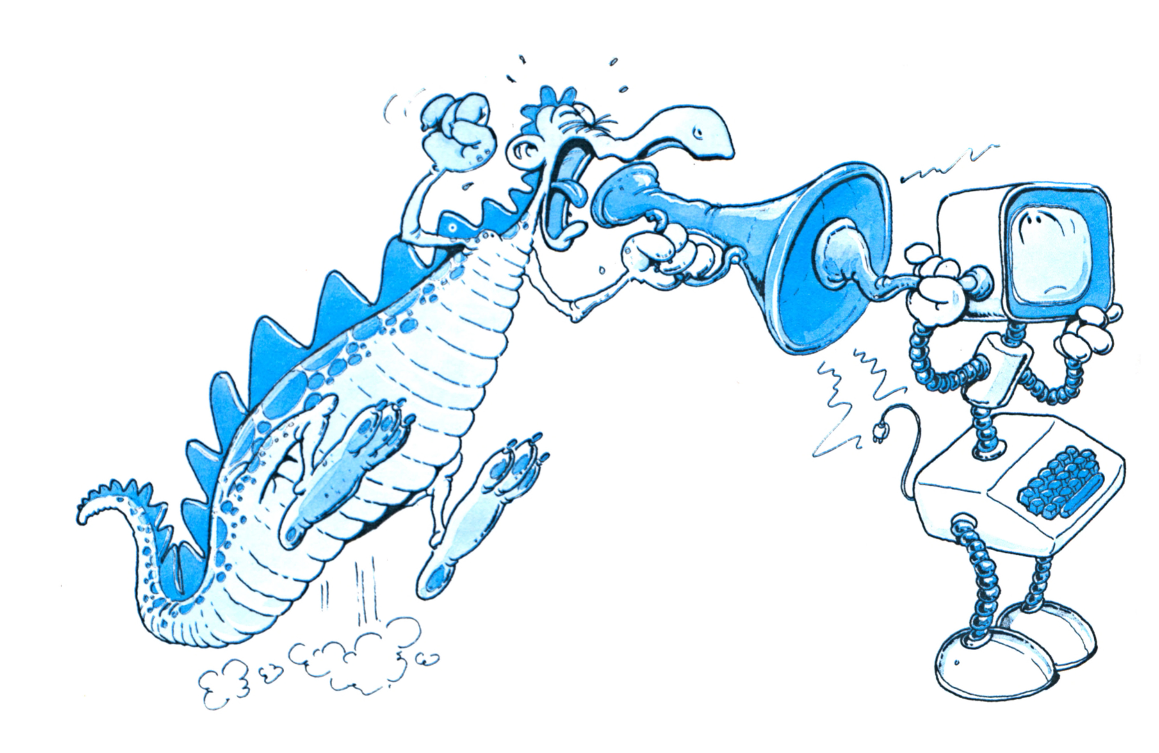
Les ordinateurs sont rapides, rigoureux et persévérants. Ils ne font que ce qu'on leur commande mais le font avec précision. Pour communiquer avec votre ordinateur, vous devez vous-même être précis. Si vous faites la moindre erreur en écrivant une instruction BASIC, certes votre ordinateur n'en subira aucun dommage, mais votre programme ne sera pas exécuté convenablement. L'écran affichera une «erreur de syntaxe» (*Syntax error*).

Rappelons que la *syntaxe* est ici l'ensemble des règles qui définissent la façon d'écrire correctement une instruction BASIC. Les règles de syntaxe sont impératives et n'admettent pas la moindre dérogation. (Aussi ne peut-on se permettre d'utiliser une orthographe approximative comme de mettre une virgule là où un point est prescrit.) Chaque caractère est rigoureusement interprété par l'ordinateur et a une signification précise. Toute transgression du code conduit à l'échec ou du moins à des résultats inattendus.

Les règles sont simples, directes et, lors de l'élaboration d'un programme, faciles à suivre. Réservez votre créativité pour la conception générale du programme et suivez très attentivement les instructions et les recommandations qui vous seront données ici.



«Attention : soyez précis !»



2

COMMUNIQUER AVEC VOTRE ORDINATEUR

Dans ce chapitre, vous allez apprendre à communiquer avec votre ordinateur. Vous lui donnerez des instructions en BASIC et il affichera des mots et des phrases. Il y aura donc échange d'informations. Cet échange portera sur des programmes (les instructions en BASIC que vous envoyez) et des données (les nombres et les caractères que vous envoyez ou que vous recevez).

On vous apprendra comment utiliser le clavier d'un ordinateur afin d'envoyer les instructions, et en particulier, comment déplacer le curseur sur l'écran et corriger les erreurs de frappe. Vous donnerez ensuite vos premières instructions en BASIC et recevrez de l'ordinateur des messages affichés sur l'écran. Vous apprendrez la différence entre exécution *immédiate* et exécution *différée.* Enfin, on vous indiquera les différentes étapes nécessaires pour écrire et exécuter un programme simple.

A la fin de ce chapitre, vous serez familiarisé avec les techniques de base qui vous sont indispensables pour communiquer avec votre ordinateur.

Utilisation du clavier

Le clavier de l'Amstrad ressemble à celui d'une machine à écrire, mais comporte quelques touches supplémentaires.

Un clavier d'ordinateur.



Les principales touches sont :

1. les lettres de l'alphabet (A-Z)
2. les chiffres de 0 à 9
3. les symboles tels que = , + , *, '', et $
4. une touche retour-chariot, portant l'indication ENTER
5. deux touches SHIFT, une touche contrôle (CTRL), une touche TAB.
6. une touche DEL (effacement) et une touche CAPS LOCK.

L'Amstrad dispose aussi d'un bloc de touches numériques séparé.

Familiarisez-vous avec le clavier.



Caractères, Nombres et Symboles

La fonction de chaque *caractère,* de chaque *nombre* et de chaque *touche* portant un symbole spécial est évidente. Ces touches sont utilisées de la même façon que sur une machine à écrire courante.

Le mode initial des touches lettres est le mode minuscule.

Le retour-chariot

Sur une machine à écrire, le bras ou la touche retour-chariot remplit deux fonctions : il ramène le chariot au début d'une ligne et il fait avancer le papier jusqu'à la ligne suivante. Sur un ordinateur la touche RETOUR-CHARIOT positionne le *curseur* sur l'écran au début de la ligne suivante. (Le curseur est un carré qui indique où le prochain caractère va être affiché sur l'écran.) Sur un ordinateur, la touche retour-chariot pourrait être appelée touche d'entrée puisque sa fonction principale est d'entrer un caractère ou une ligne d'un texte ou encore des données dans la mémoire de l'ordinateur. Et en fait, sur certains claviers, cette touche est maintenant marquée ENTER (entrée) ; sur d'autres elle est marquée d'une flèche. Sur l'Amstrad cette touche est appelée ENTER.

Chaque instruction entrée dans l'ordinateur, y compris les instructions BASIC, doit être suivie d'un retour-chariot. Il signifie «Entrez la ligne dans la mémoire». Tout ce que l'on tape sur une ligne est ignoré par l'ordinateur jusqu'à ce que l'on ait enfoncé la touche RETOUR-CHARIOT. On peut ainsi corriger les erreurs avant d'entrer la ligne dans la mémoire de l'ordinateur.

Le RETOUR est utilisé seulement lors de la frappe. Bien que très utile, cette commande n'est pas stockée comme une partie d'instruction d'un programme. Dans ce chapitre d'introduction, nous allons vous indiquer tous les caractères que vous devez frapper,

et pour vous faciliter les choses nous représenterons le RETOUR à l'aide du symbole ↵ à la fin de chaque ligne. **Souvenez-vous** que vous devez enfoncer la touche ENTER pour transmettre une instruction à l'ordinateur ; si vous ne le faites pas, il ne se passera rien. De la même façon, chaque fois que l'ordinateur vous pose des questions, il faut frapper la touche ENTER pour transmettre la réponse.

Shift

La touche SHIFT fonctionne de la même façon que la touche majuscule sur une machine à écrire. Elle permet de sélectionner les symboles inscrits soit à la partie inférieure, soit à la partie supérieure des touches. Si vous ne pressez pas la touche SHIFT, c'est le symbole inférieur de la touche enfoncée qui est généré. Si vous tapez la touche SHIFT, le symbole supérieur de la touche apparaît sur l'écran.

Contrôle

La touche CONTROLE (ou CTRL) permet de communiquer à l'ordinateur des commandes abrégées fréquemment utilisées. Elle n'existe pas sur les machines à écrire et s'utilise de la même façon que la touche SHIFT : il faut l'enfoncer et la maintenir enfoncée tout en appuyant sur une autre touche du clavier. Cette opération permet de générer un caractère de contrôle. C'est une méthode pratique pour obtenir une commande ou un code de contrôle puisqu'il suffit d'appuyer sur deux touches.

(Les *codes de contrôle* facilitent l'utilisation de commandes usuelles telles que le déplacement du curseur sur l'écran vers le bas, vers le haut, vers la droite, vers la gauche. Conformément à l'instruction qui lui a été donnée, le curseur peut se déplacer d'une position sur la position la plus proche, ou passer au mot voisin ou à la ligne suivante. L'utilisation des caractères de contrôle est spécifique à chacun des programmes fournis à votre ordinateur et la documentation relative à ces programmes vous donnera toutes les explications concernant ces caractères. Ici nous n'utiliserons pas de code de contrôle.

TAB

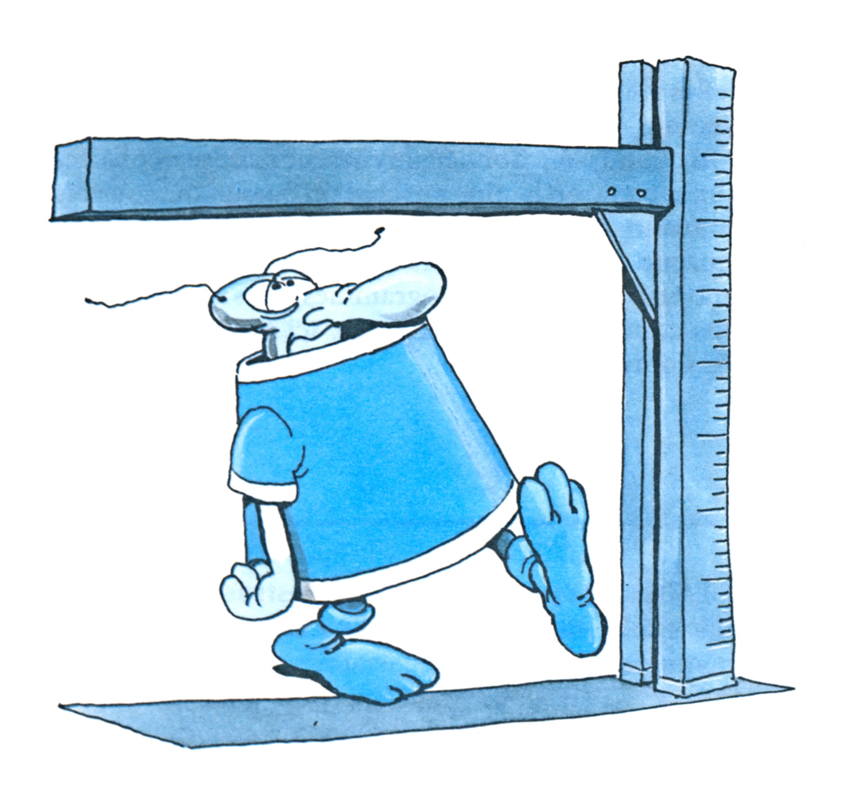
La touche TAB est utilisée dans certaines instructions pour afficher des messages à des endroits précis de l'écran.

CAPS LOCK

Cette touche est utilisée pour bloquer le clavier en mode majuscule. Pour revenir au mode minuscule il suffit de la taper à nouveau.

DEL

Cette touche est utilisée pour effacer un caractère. Lorsque vous tapez cette touche, le curseur efface le caractère qui le précède et prend sa place.



Le curseur

Rappelons que le curseur est un carré sur l'écran, qui indique la position du prochain caractère affiché. Voici un curseur qui vous indique l'endroit où vous vous trouvez une fois que vous avez tapé BONJOUR :

```
BONJOUR ■
```

En déplaçant le curseur horizontalement et verticalement sur l'écran vous pouvez remplacer aisément les caractères affichés et ainsi modifier le texte que vous venez de taper. Il s'agit là d'une méthode de correction très pratique et vous serez sans aucun doute amené à l'utiliser de manière intensive pour corriger vos erreurs de frappe.


Apprenez à déplacer le curseur sur l'écran.

Le clavier de l'Amstrad dispose de 4 touches permettant de déplacer le curseur sur l'écran (↑ , ← , → , ↓).

Enfin la touche DEL déplace le curseur vers la gauche en effaçant les caractères.

Peut-être pensez-vous qu'il est temps de retourner à votre clavier et de mettre en pratique ce que vous venez d'apprendre. Quelques exercices au clavier vous permettront de vous familiariser avec

ce que nous avons vu. Puis, nous apprendrons à écrire les instructions BASIC pour l'ordinateur.

Parler BASIC

Lorsque vous branchez votre Amstrad, l'interpréteur BASIC est automatiquement activé et l'écran affiche :

```
Amstrad 64K Microcomputer (v1)

© 1984 Amstrad Consumer Electronics plc
          and locomotive Software Ltd

BASIC 1.Ø

Ready
■
```

Amstrad est le nom donné à l'ordinateur, BASIC 1.0 désigne la version. Le mot Ready est l'indicatif ou le message guide opérateur. Ce message signifie : «Je suis prêt, dites-moi ce que je dois faire.» Dès que ce symbole apparaît, l'ordinateur est prêt à recevoir vos instructions.

Nous allons poursuivre en supposant que :

1) l'interpréteur est prêt à fonctionner ;
2) le mot Ready est bien affiché en haut à gauche de l'écran. Si Ready n'apparaît pas, débranchez votre ordinateur et remettez-le sous tension.

Nous allons maintenant entrer notre première instruction BASIC dans l'ordinateur. Tapez l'instruction suivante (exactement telle qu'elle apparaît ici) :

```
PRINT"BONJOUR"
```

Sur l'écran vous devez voir :

```
Ready
PRINT"BONJOUR" ■
```

Le mot Ready sur la gauche est l'*indicatif* qui signifie que l'interpréteur BASIC attend une instruction. Vous venez juste de taper les caractères situés en dessous. Rien ne devrait se passer. Vous souvenez-vous pourquoi ?

Tout simplement parce que vous devez enfoncer la touche ENTER pour entrer votre instruction. Appuyez maintenant sur la touche ENTER. Cette fois-ci vous pouvez voir :

```
PRINT"BONJOUR"
BONJOUR
Ready
■
```

L'interpréteur a reçu votre instruction et l'a immédiatement exécutée en affichant BONJOUR comme vous l'avez demandé. Puis l'interpréteur affiche un nouvel indicatif Ready vous indiquant qu'il est prêt à recevoir une nouvelle instruction.

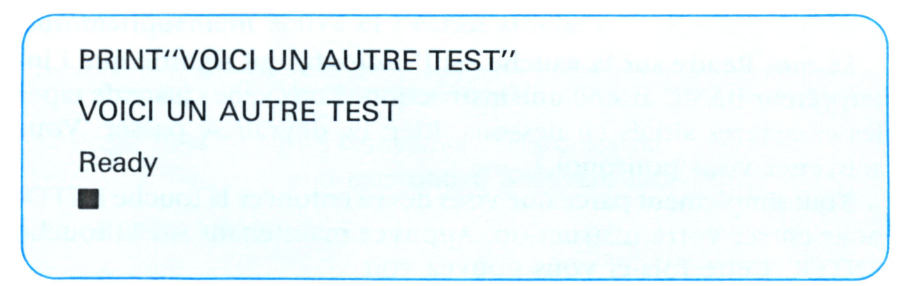
Examinons de plus près notre première instruction BASIC :

```
PRINT"BONJOUR"
```

Cette instruction comprend deux parties : PRINT et «BONJOUR». PRINT est un *mot réservé* ayant pour l'interpréteur une signification bien définie. «BONJOUR» est le message à imprimer et doit être inséré entre guillemets. Vous pouvez écrire n'importe quel message entre les guillemets. Voyons un autre exemple. Tapez :

PRINT"VOICI UN AUTRE TEST"

et vous voyez sur l'écran :



Essayez à nouveau. Affichez votre propre message, mais attention : si vous oubliez un des guillemets ou si vous orthographiez mal le mot PRINT, l'instruction ne sera pas exécutée, et vous obtiendrez un message d'erreur. Continuez. Faites autant d'essais que vous voudrez. Vous ne risquez pas d'endommager l'appareil.

Vous vous demandez sans doute pourquoi cette instruction s'appelle PRINT (imprimer) puisqu'elle ne fait qu'afficher sur l'écran et n'imprime pas l'information. Même si vous disposez d'une imprimante connectée à votre ordinateur, rien ne sera imprimé. Le choix du mot PRINT tient à ce que les premiers terminaux ressemblaient à des machines à écrire et en fait imprimaient l'information au lieu de l'afficher sur un écran. La technologie a changé, mais l'instruction BASIC PRINT est demeurée inchangée. A présent, pour envoyer l'information à l'imprimante et non à l'écran de visualisation, on utilise une autre instruction.

Avant de poursuivre assurez-vous que l'indicatif BASIC réapparaît bien sur l'écran, c'est-à-dire que l'interpréteur BASIC est prêt à accepter une autre instruction. S'il n'est pas affiché, vous ne pourrez pas taper de nouvelle commande. Essayez alors d'enfoncer les touches ENTER ou SHIFT CTRL et ESC. Si rien ne se produit, débranchez l'ordinateur, remettez-le sous tension et recommencez l'opération.

Nous allons maintenant écrire non plus une simple instruction mais notre premier programme BASIC. Tapez les trois lignes suivantes :

```
10 PRINT"BONJOUR" ↵
20 PRINT"COMMENT ALLEZ-VOUS ?" ↵
30 END ↵
```

L'écran doit afficher :

```
10 PRINT"BONJOUR"
20 PRINT"COMMENT ALLEZ VOUS?"
30 END
■
```

Rien ne se passe et cela peut vous surprendre. L'ordinateur répond simplement par le curseur. Vous venez d'écrire trois instructions successives. Mais en réalité vous avez écrit un petit *programme* BASIC. Notons que chaque ligne commence par un numéro appelé numéro de ligne ou *étiquette*. Il indique à l'ordinateur que nous voulons écrire un programme complet et non exécuter chaque instruction immédiatement. Tapez :

```
RUN ↵
```

L'écran affiche maintenant :

```
BONJOUR
COMMENT ALLEZ-VOUS?
Ready
■
```



«Prêt ? Partez !»

Qu'avons-nous fait ? Nous avons d'abord créé un programme de trois lignes et nous l'avons stocké en mémoire, ligne par ligne, en appuyant chaque fois sur la touche ENTER. Nous avons ensuite exécuté le programme en tapant la commande RUN. Telle est la méthode normale d'écriture et d'exécution d'un programme. Nous allons désormais suivre cette procédure. Toutes les lignes seront précédées d'une étiquette et le programme sera exécuté automatiquement suivant l'ordre croissant des étiquettes.

S'il vous arrive de taper une instruction BASIC sans étiquette à la suite du mot Ready, elle sera exécutée immédiatement mais ne sera pas mémorisée. C'est ce que l'on nomme, en BASIC, le *mode immédiat* ou *mode calculateur.* En revanche, si vous tapez une instruction BASIC précédée d'une étiquette, elle sera mémorisée quand vous taperez la touche ENTER et ne sera exécutée que lorsque vous aurez tapé la commande RUN. C'est le *mode différé* ou mode normal. Voyons leur fonctionnement. Notre programme de

trois lignes a été stocké en mémoire. Il peut être exécuté indéfiniment, étendu ou modifié. Maintenant tapez :

RUN ↵

Vous voyez à nouveau s'afficher sur l'écran :

```
BONJOUR
COMMENT ALLEZ-VOUS?
Ready
■
```

Examinons maintenant la mémoire de l'ordinateur et affichons son contenu. Pour cela tapez :

LIST ↵

Sur l'écran apparaît :

```
10  PRINT"BONJOUR"
20  PRINT"COMMENT ALLEZ-VOUS?"
30  END
Ready
■
```

Votre programme est «listé» sur l'écran tel qu'il se trouve dans la mémoire de l'ordinateur.

Pour bien comprendre la différence entre l'instruction immédiate et l'instruction différée, tapez :

PRINT"AU REVOIR" ↵

Sur l'écran vous voyez :

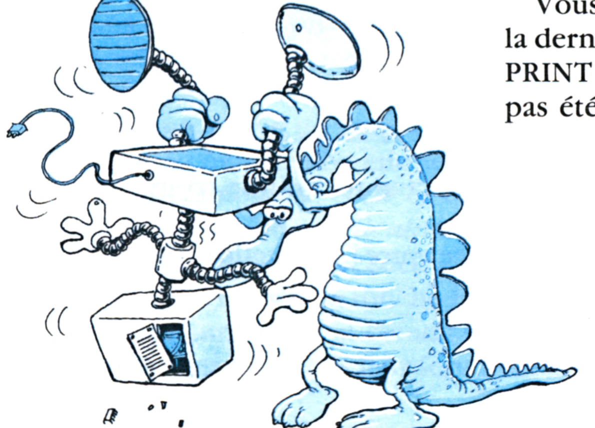
```
PRINT"AU REVOIR"
AU REVOIR
Ready
■
```

Maintenant, tapez :

LIST ↵

à nouveau, et vous voyez sur l'écran :

```
10  PRINT"BONJOUR"
20  PRINT"COMMENT ALLEZ-VOUS?"
30  END
Ready
■
```



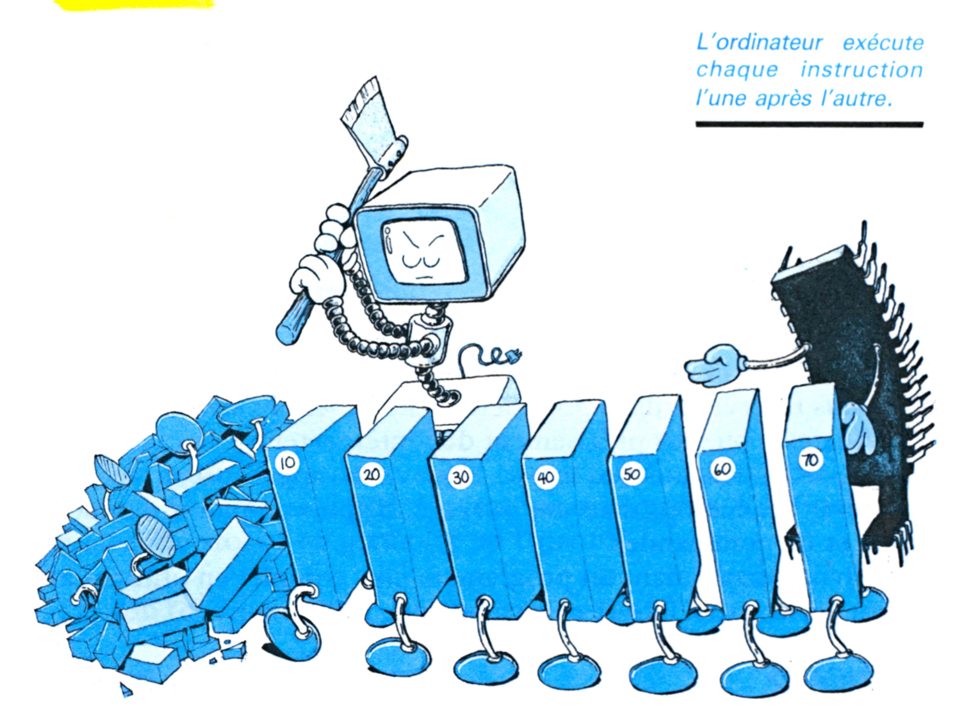
Vous ne retrouvez nulle part la dernière instruction donnée : PRINT «AU REVOIR». Elle n'a pas été mémorisée !

Si vous n'avez pas utilisé de numéro de ligne, votre instruction est perdue !

En résumé, une instruction immédiate est exécutée dès que vous la tapez. Elle n'est pas stockée dans la mémoire de l'ordinateur ; il faut donc la taper chaque fois que vous voulez qu'elle soit exécutée. Cette méthode s'appelle *mode d'exécution immédiat*. En pratique, ce mode est peu utilisé ; on s'en servira pour vérifier certaines valeurs une fois le programme arrêté. N'hésitez pas à taper librement des instructions BASIC afin d'observer ce qui se passe.

Quand une ligne est précédée d'une étiquette, le mode s'appelle *mode d'exécution différé*. Dans ce cas, chaque ligne du programme est stockée dans la mémoire de l'ordinateur. Quand vous avez achevé d'entrer un programme, il vous suffit de taper la commande RUN pour qu'il soit exécuté. Répétons que le contenu de la mémoire vive (RAM) disparaît, quel que soit le programme entré, si vous débranchez l'ordinateur. Si vous souhaitez conserver votre programme, vous devez le sauvegarder sur une cassette ou sur une disquette.

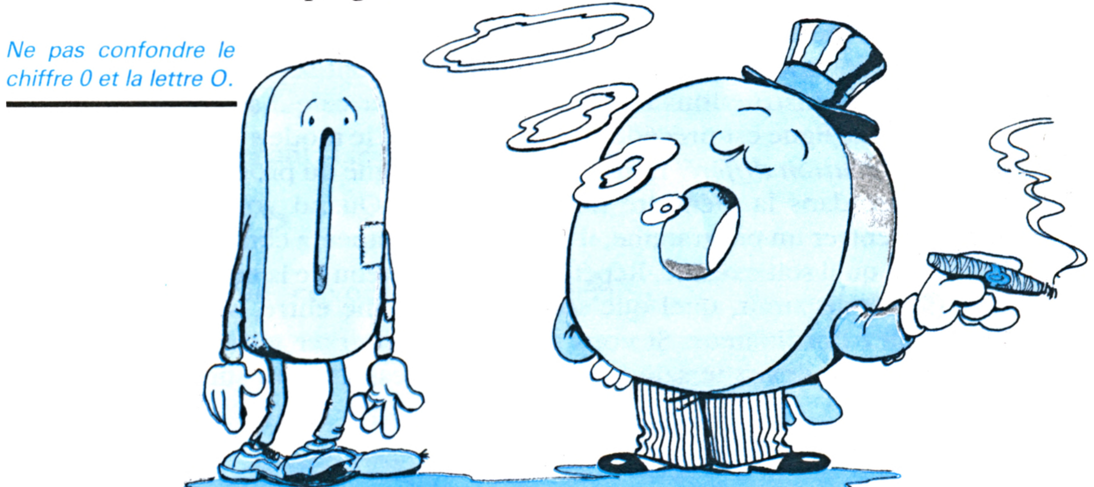
L'ordinateur exécute chaque instruction l'une après l'autre.



Chaque ligne d'un programme BASIC doit commencer par une étiquette. L'ordre croissant des numéros définit l'ordre d'exécution des lignes. Ainsi, la ligne 10 sera exécutée avant la ligne 20.

La commande END, ligne 30 dans notre exemple, est facultative. Elle informe l'interpréteur qu'il a atteint la FIN normale et non accidentelle du programme.

Ne pas confondre le chiffre 0 et la lettre O.



Notons enfin que le chiffre 0 doit se différencier de la lettre O. Pour éviter toute confusion, on a pris l'habitude de représenter le chiffre 0 par un O barré. Le zéro est représenté ainsi sur le clavier de l'Amstrad.

Un programme plus long

RUN (marche) et LIST (liste) s'appellent des *commandes*. Ce sont des mots réservés utilisés pour définir des fonctions spécifiques de l'ordinateur, ou plus précisément de l'interpréteur. Ces commandes appartiennent aussi au BASIC. Nous allons apprendre, à mesure que nous avancerons dans cet ouvrage, de plus en plus d'instructions et de commandes.

Ecrivons maintenant un programme plus complet en utilisant à la fois l'instruction PRINT et la commande NEW (nouveau). Tapez :

```
NEW ↵
```

Puis :

LIST ↲

Il ne se passe rien. En effet la mémoire ne contient plus de programme. La commande NEW efface la mémoire de l'ordinateur. Maintenant tapez :

```
NEW
10 PRINT"VOICI" ↲
20 PRINT"UN AUTRE" ↲
30 PRINT"EXEMPLE" ↲
40 END ↲
```

Découvrons ensemble l'objet de ce programme. Tapez :

RUN ↲

Vous voyez maintenant sur l'écran :

```
VOICI
UN AUTRE
EXEMPLE
Ready
■
```

Notre programme affiche le texte comme prévu. Vérifions qu'il a bien été stocké en mémoire en tapant :

LIST ↲

Nous voyons sur l'écran :

```
LIST

10  PRINT"VOICI"
20  PRINT"UN AUTRE"
30  PRINT"EXEMPLE"
40  END
Ready
■
```

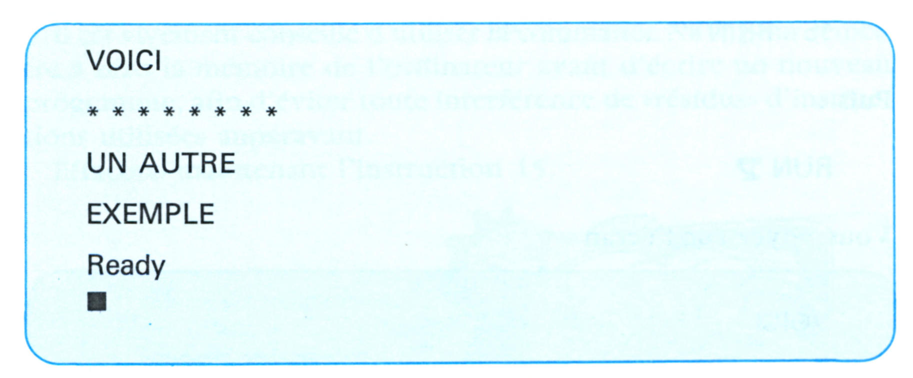
La commande NEW a été utilisée pour effacer la mémoire de l'ordinateur, c'est-à-dire pour faire de la place à un nouveau programme. Si la commande NEW n'est pas utilisée, les nouvelles instructions effacent et remplacent celles qui portaient précédemment la même étiquette. Ceci peut être source d'erreurs puisque des «résidus» d'instructions antérieures peuvent «se retrouver» dans le nouveau programme. Si vous n'utilisez pas la commande NEW, à chaque fois que vous entrez une instruction comportant l'étiquette 10, par exemple, sa nouvelle version effacera automatiquement l'ancienne. Mais *attention* : si vous avez précédemment écrit une instruction comportant l'étiquette 15, et si vous entrez le programme ci-dessus sans avoir auparavant tapé NEW, l'instruction 15, restée dans la mémoire de l'ordinateur, sera automatiquement incorporée à votre nouveau programme (puisque l'étiquette 15 n'est pas utilisée dans celui-ci). En voici la démonstration. Tapez :

```
15 PRINT "* * * * * * * * *" ↵
```

Puis :

```
RUN ↵
```

Nous avons sur l'écran :



Comme vous pouvez le constater, l'instruction PRINT comportant l'étiquette 15 est automatiquement insérée dans le programme entre les instructions 10 et 20. Listons ce programme. Tapez :

LIST ↵

Vous voyez sur l'écran :

```
10  PRINT"VOICI"
15  PRINT"* * * * * * * *"
20  PRINT"UN AUTRE"
30  PRINT"EXEMPLE"
40  END
Ready
■
```

L'instruction 15 fait maintenant partie du programme. Chaque fois que vous entrez une instruction comportant une étiquette, elle est automatiquement insérée, dans le bon ordre, dans la mémoire de l'ordinateur.

Nous avons dit que, lorsque l'on utilise une étiquette *déjà existante,* la nouvelle instruction efface automatiquement l'instruction antérieure. Mettons cela en application pour supprimer l'instruction 15. Tapez :

```
15  PRINT". . . . . . . ."
```

Puis :

```
RUN ↵
```

Vous voyez sur l'écran :

```
VOICI
. . . . . . . . .
UN AUTRE
EXEMPLE
Ready
■
```

Comme vous pouvez le constater, la nouvelle instruction PRINT comportant l'étiquette 15 a remplacé l'instruction antérieure. Vérifions-le en tapant :

```
LIST ↵
```

Vous voyez sur l'écran :

```
LIST
10  PRINT"VOICI"
15  PRINT". . . . . . . ."
20  PRINT"UN AUTRE"
30  PRINT"EXEMPLE"
40  END
Ready
■
```

Il est vivement conseillé d'utiliser la commande NEW pour remettre à zéro la mémoire de l'ordinateur avant d'écrire un nouveau programme, afin d'éviter toute interférence de «résidus» d'instructions utilisées auparavant.

Effaçons maintenant l'instruction 15.

«Moi, c'est comme ça que j'efface.»



Nous allons taper :

15 ↵

Cette instruction est une simple étiquette. On l'appelle une *instruction vide.* Elle ne fait qu'effacer une instruction antérieure portant le même numéro. Tapons maintenant RUN. Sur l'écran apparaît :

```
VOICI
UN AUTRE
EXEMPLE
Ready
■
```

Mais attention : cette fonction peut être dangereuse. En effet, si par accident vous tapez :

```
20
```

suivi de ENTRÉE, c'est l'instruction 20 qui sera supprimée et remplacée par une «instruction vide». Donc pour éviter une mauvaise surprise, vérifiez toujours le listing du programme avant de l'exécuter.



Résumé

Nous savons maintenant écrire des programmes élémentaires en BASIC qui affichent des informations sur l'écran. Nous savons aussi écrire les instructions d'un programme BASIC comportant des étiquettes. Nous avons montré pourquoi les programmes doivent être précédés de la commande NEW et se terminer par END. Nous avons vu qu'un programme aussitôt entré au clavier est automatiquement stocké dans la mémoire de l'ordinateur, et qu'il peut être exécuté au moyen de la commande RUN. Nous avons vu enfin que l'on obtient la liste des instructions en tapant la commande LIST.

L'exécution des instructions d'un programme se fait dans l'ordre croissant des étiquettes. Si vous tapez deux fois la même étiquette, soit volontairement, soit par erreur, la nouvelle instruction efface automatiquement l'instruction antérieure portant le même numéro. Si à un moment donné vous voulez insérer dans le programme une ligne avec un nouveau numéro, l'interpréteur le fera automatiquement au bon endroit.

Dans ce chapitre, ont été introduites un certain nombre de notions nouvelles. Si vous voulez apprendre à programmer, il est indispensable que vous mettiez en pratique ce que nous venons d'expliquer. Il vous est vivement conseillé de faire les exercices qui suivent. Les réponses à ces exercices se trouvent à la fin du livre.

Exercices

2-1 : Ecrivez un programme qui imprime «BONNE JOURNEE».

2-2 : Ecrivez un programme qui permette d'engendrer la suite de lettres suivante :

```
AAAAA
BBBB
CCC
DD
E
```

2-3 : Ecrivez un programme qui imprime :

```
*******
*TITRE*
*******
```

2-4 : Donnez la définition des termes suivants :

a - étiquette

b - exécution différée

c - exécution immédiate

d - instruction vide

e - curseur

f - mot réservé

g - indicatif (message guide opérateur)

2-5 : Pourquoi l'instruction PRINT est-elle utilisée pour afficher sur l'écran ?

2-6 : Pouvez-vous exécuter un programme en tapant les instructions une par une en mode immédiat ?

2-7 : Pourquoi faut-il utiliser la commande NEW avant de taper un nouveau programme ?

2-8 : Pouvez-vous taper dans le désordre des instructions comportant une étiquette ?

2-9 : L'instruction suivante est-elle une façon correcte d'afficher le mot EXEMPLE :

```
PRINT EXEMPLE
```

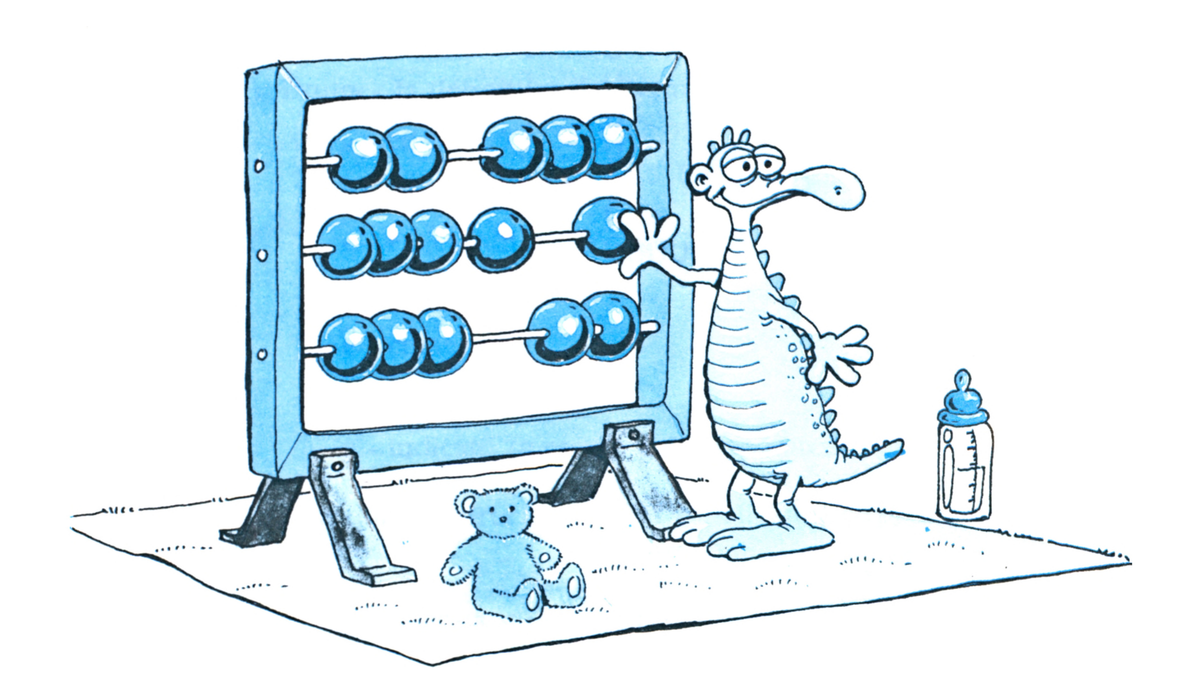
2-10 : A quoi sert la touche ENTER ?

2-11 : Comment effacer l'instruction 20 d'un programme ?

2-12 : Si vous avez déjà tapé l'instruction 30 et que vous souhaitiez la remplacer par une autre instruction 30, devez-vous commencer par effacer la première ?

2-13 : Ecrivez un programme qui affiche ce qui suit :

```
OOOOO     U   U    I
O   O     U   U    I
O   O     U   U    I
O   O     U   U    I
O   O     U   U    I
O   O     U   U    I
OOOOO     UUUUU    I
```



3

LES CALCULS EN BASIC

Dans ce chapitre, nous allons commencer à travailler sur les nombres, c'est-à-dire les afficher, faire des additions, des soustractions, des multiplications et des divisions. Nous apprendrons à faire des calculs en utilisant des opérateurs arithmétiques et enfin, nous décrirons les autres opérateurs importants propres au BASIC.

Impression des nombres

Jusqu'ici nous avons uniquement affiché du texte. Affichons maintenant un nombre. Tapez :

```
PRINT 3 ↵
```

L'écran doit afficher :

```
3
```

D'après ce que nous avons vu au Chapitre 2, nous pouvons dire que cette instruction est en mode immédiat : elle ne comporte pas d'étiquette et est exécutée immédiatement. Dans ce chapitre, tous nos exemples seront en mode immédiat, et vous pourrez ainsi les exécuter en appuyant simplement sur quelques touches.

Notons qu'en BASIC seul le texte est inséré entre guillemets. Pour les chiffres, les guillemets sont superflus. Le texte délimité par des guillemets s'appelle une *chaîne* qui peut bien sûr comporter des nombres.

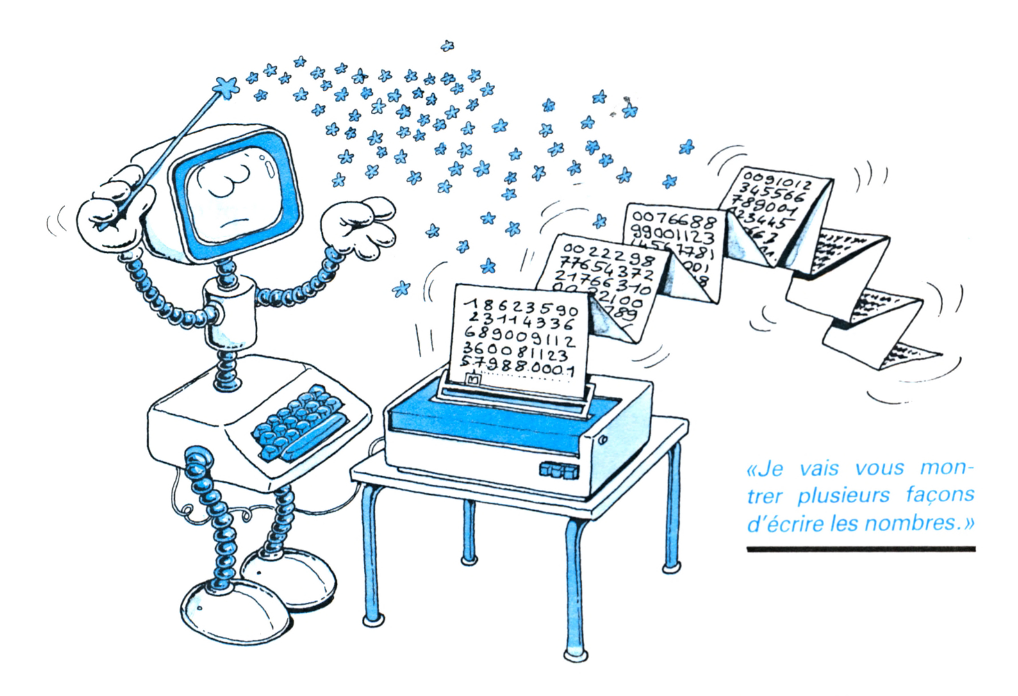
Essayons d'imprimer de grands nombres : 100, 1000, 10000, etc. A un moment donné, l'interpréteur BASIC va refuser d'imprimer et va afficher le message Overflow error (erreur). La raison en est simple : chaque interpréteur BASIC est prévu pour traiter les nombres entiers jusqu'à une valeur limite.

Votre interpréteur BASIC accepte aussi les nombres décimaux. Tapez :

```
PRINT 1.5 ↵
```

Votre écran affiche 1.5.

Dans le jargon informatique, les nombres décimaux s'appellent *nombres à virgule flottante.* Notez que, selon l'usage en anglais, la virgule décimale est représentée par un point.



«Je vais vous montrer plusieurs façons d'écrire les nombres.»

Notation scientifique

Voyons les nombres décimaux de plus près. Comme pour les nombres entiers, l'interpréteur dans le cas des nombres décimaux limite le nombre de chiffres qu'il retiendra. Par exemple : la valeur d'1/3 est :

0.333333... (etc.)

cette valeur sera stockée dans l'ordinateur sous la forme :

0.333333333 *(9 chiffres significatifs après la virgule)*

On dit que la valeur réelle du nombre a été *tronquée* (coupée) à 9 chiffres. (On a donc une approximation, généralement suffisante.)

Votre interpréteur BASIC traite les nombres décimaux, il utilise également une représentation scientifique des nombres. Quand un nombre est trop grand ou trop petit, il est affiché en *notation scientifique*, ce qui prend moins de place. Voici un exemple :

3.2E6

qui signifie :

$3.2 \times 10^6 = 3\ 200\ 000$

10^6 signifie 10 à la puissance 6 c'est-à-dire 10 multiplié cinq fois par lui-même :

$10 \times 10 \times 10 \times 10 \times 10 \times 10 = 1\ 000\ 000$

De même :

1.12E − 7

signifie :

$1.12 \times 10^{-7} = 0.000000112$

10^{-7} signifie 1/10 à la puissance 7 c'est-à-dire 1/10 multiplié six fois par lui-même ($1/10 = 10^{-1}$).

Calculs arithmétiques

Effectuons quelques calculs arithmétiques simples. Tapez :

PRINT 2 + 2↵

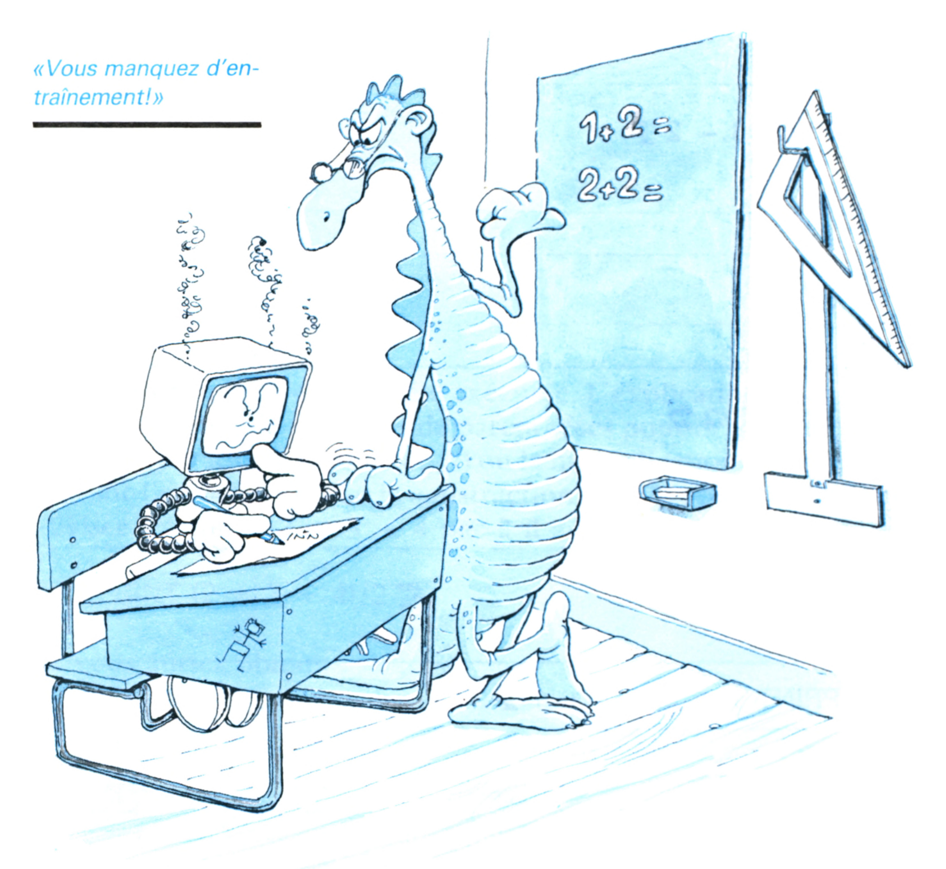
Le résultat sur votre écran est :

```
4
```

Nous venons de faire notre premier calcul arithmétique. Le symbole de l'addition (+) s'appelle un *opérateur.* Il représente une opération à réaliser sur un ou plusieurs *opérandes.* Le BASIC comporte cinq opérateurs arithmétiques :

+	(*plus*)
—	(*moins*)
*	(*multiplication*)
/	(*division*)
↑	(*exponentiation ou puissance*)

«Vous manquez d'entraînement!»



Essayons à présent ceci. Tapez :

```
PRINT 2 * 3 ↵
```

Le résultat est 6. Le symbole * est le symbole de la multiplication. On a pris l'habitude d'utiliser ce symbole au lieu du symbole tra-

ditionnel × pour éviter toute confusion avec la lettre X. Voici d'autres exemples corrects d'instructions arithmétiques :

```
PRINT 1 + 2 * 3 ↵
```

Vous obtenez :

```
7
```

```
PRINT 3 − 2 ↵
```

Résultat :

```
1
```

```
PRINT 8/2 ↵
```

Résultat :

```
4
```

```
PRINT 1 + 2 + 3 + 4 ↵
```

Résultat :

```
10
```

Votre BASIC traitant les nombres décimaux, l'instruction suivante est, elle aussi, acceptable :

```
PRINT (6/3 + 12/4)/2 ↵
```

Résultat :

```
2.5
```

Notons que les parenthèses ont été utilisées dans ce cas pour bien distinguer les différents groupes d'opérations. Essayons un autre exemple. Tapez :

PRINT 2 + 3 + 4/2 ↵

Résultat :

```
7
```

Ici la division 4/2 a d'abord été exécutée. L'explication en est simple : en BASIC, s'il n'y a pas de parenthèses pour déterminer l'ordre des opérations, la division (/) ou la multiplication (*) ont priorité sur l'addition (+) ou la soustraction (—). Si vous aviez voulu diviser le groupe 2 + 3 + 4 par 2 il aurait fallu taper :

PRINT (2 + 3 + 4)/2 ↵

et vous auriez obtenu :

```
4.5
```

Il est bon d'utiliser largement les parenthèses pour éviter toute confusion. Par exemple, l'expression suivante (ou groupe de valeurs et d'opérateurs) :

$$\frac{1 + 2 + 3}{4 + 5} \times 3$$

peut s'écrire en BASIC de la façon suivante :

```
((1 + 2 + 3)/(4 + 5)) * 3
```

ou bien :

```
(1 + 2 + 3)/(4 + 5) * 3
```

car le calcul se fait de la gauche vers la droite quand les opérateurs sont également prioritaires, c'est-à-dire qu'ici la division est effectuée avant la multiplication. En résumé : utilisez des parenthèses pour isoler les groupes. Quand vous ouvrez une parenthèse ayez soin de la refermer.

Formats d'affichage

Si vous tapez :

```
PRINT"2 FOIS TROIS=";2*3 ↵
```

vous obtiendrez :

```
2 FOIS TROIS= 6
```

Dans cette instruction PRINT nous venons d'utiliser un texte et des nombres, séparés par un point-virgule. Plus précisément, nous avons utilisé une *expression,* 2*3, plutôt qu'un nombre. Maintenant tapez :

```
PRINT"2 FOIS TROIS =;2*3" ↵
```

et vous obtenez :

```
2 FOIS TROIS=;2*3
```

Vous avez là une instruction acceptable, mais pas celle que vous attendiez. Répétons-le : tout ce qui est entre guillemets est affiché littéralement. Pour obtenir le résultat voulu, il faut que la virgule ou le point-virgule soient situés à *l'extérieur des guillemets.*

L'instruction PRINT peut être utilisée pour imprimer plusieurs éléments sur une même ligne. Toutefois, ils doivent être séparés par une virgule ou un point-virgule. Si vous utilisez une virgule, vous obtiendrez un plus grand espace entre les éléments, l'espace généré par le point-virgule étant plus petit. Tout comme une touche de tabulation sur machine à écrire, la virgule est utilisée pour créer des *tabulations,* c'est-à-dire des champs sur l'écran. Cette technique est pratique pour afficher des tableaux. Voyons cette nouvelle fonction. Tapez :

```
PRINT1;2;3 ↵
```

Résultat :

```
1 2 3
```

Maintenant tapez :

```
PRINT 1,2,3 ↵
```

Cette fois-ci vous obtenez :

```
1            2            3
```

Calculons maintenant la T.V.A. sur une vente s'élevant à 1234 F. Le taux de la TVA est de 18,6 %. L'instruction est :

```
PRINT"T.V.A. =";1234*18.6/100 ↵
```

Vous obtenez :

```
T.V.A. = 229.524
```

Nous pouvons aussi taper :

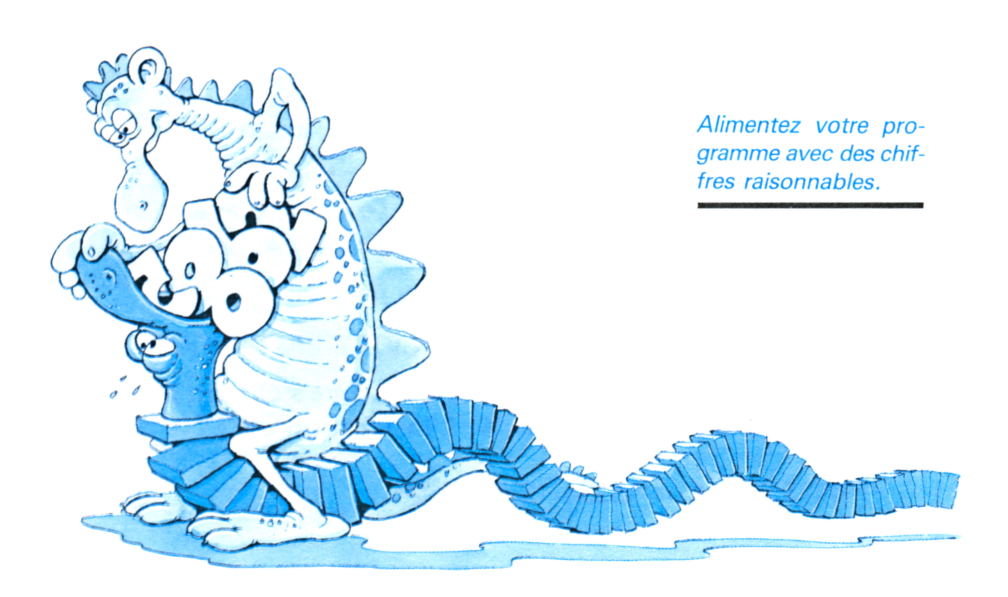
```
PRINT"T.V.A. =";1234*0.186
```

et nous obtenons le même résultat. En fait il existe plusieurs façons équivalentes d'écrire un programme.

Vous pouvez imprimer de nombreux éléments sur une même ligne. Voyez plutôt :

```
PRINT1;2;3;4;5;6;7;8;9;"NOMBRES"
```

L'écran affiche :



Alimentez votre programme avec des chiffres raisonnables.

123456789 NOMBRES

Nous venons d'apprendre à effectuer des calculs arithmétiques simples et à en afficher les résultats. Mettons à profit nos connaissances pour résoudre quelques problèmes élémentaires.

Exemples d'application

Calculons la consommation d'une voiture en litres aux 100 kilomètres. La formule mathématique est :

CONSOMMATION = QUANTITE D'ESSENCE/DISTANCE EN KM * 100

Supposons que la distance soit de 510 km et la consommation de 20,2 litres. L'instruction BASIC est :

```
PRINT"LA CONSOMMATION EST DE";20.2/510*100;
"L/100 KM" ↵
```

Voici un autre exemple simple. Soit une température en Fahrenheit, l'équivalent en Celsius est obtenu par la formule :

Valeur Celsius = (valeur F — 32) × 5/9

Pour calculer l'équivalent en Celsius de 79°F, tapez :

```
PRINT"79 DEGRES F=";(79—32)*5/9;"DEGRES C" ↵
```

Vous obtenez :

```
79 DEGRES F = 26.1111 DEGRES C
```

Notons que * 5/9 n'est pas entre parenthèses, puisque l'ordre dans lequel sont exécutées la multiplication et la division est indifférent, toutes deux ayant le même niveau hiérarchique.

Résumé

Dans ce chapitre, nous avons appris à effectuer des calculs arithmétiques et à afficher le texte et les résultats sur une même ligne. Nous avons ensuite mis en application ces nouvelles connaissances pour automatiser le calcul de formules simples en écrivant des instructions BASIC sur une même ligne.

Jusqu'à présent, nous avons spécifié toutes les valeurs à l'intérieur de l'instruction BASIC. Ce que nous voulons maintenant, c'est écrire un programme où nous pourrons introduire des valeurs qui ne seront plus, comme précédemment, définies, mais variables. Ces valeurs seront représentées par ce que l'on appellera des *variables* ; et l'étude des variables fera l'objet du prochain chapitre.



Exercices

3-1 : Ecrivez une instruction BASIC qui calcule :

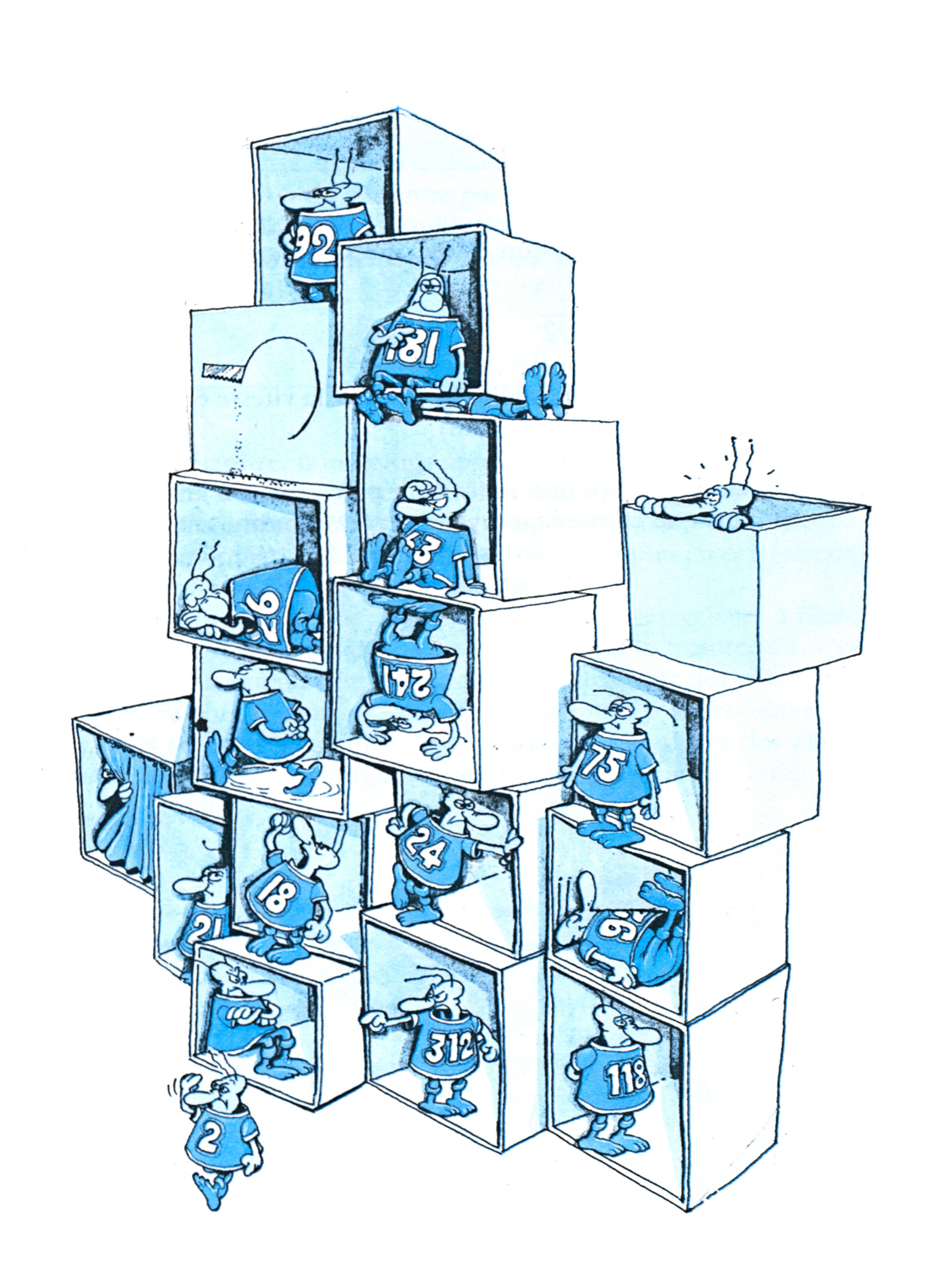
$$\frac{5 + 6}{1 + 2} : 3$$

3-2 : Ecrivez une instruction BASIC qui calcule :

$$1 + 1/2 \frac{1}{1 + 1/2}$$

3-3 : Soit une vitesse de 100 km/h, calculez la vitesse équivalente en m.p.h (1 mile = 1,6 km)

3-4 : Calculez le temps nécessaire pour parcourir 350 km, en supposant que la vitesse moyenne est 95 km/h.


92
181
75
24
18
312
118
2

4

MEMORISER DES VALEURS ET UTILISER DES VARIABLES

Dans ce chapitre, nous allons apprendre à écrire des programmes que l'on peut utiliser de façon répétée sans avoir à les modifier. Ils afficheront des résultats différents selon les données entrées au clavier. Jusqu'à présent, pour obtenir le résultat d'un calcul arithmétique tel que 2 + 3 par exemple, nous avons inclus dans le programme les nombres servant au calcul. Nous écrirons maintenant des programmes fonctionnant avec des données chaque fois différentes. Les opérations à réaliser seront définies dans les programmes. L'utilisateur entrera les données au clavier au moment de l'exécution du programme. Ainsi, le programme sera indéfiniment utilisable.

De plus, nous introduirons le concept de variable et apprendrons à utiliser deux nouvelles instructions : INPUT (entrer) et LET (soit).

Commençons par apprendre à fournir des données au programme pendant son exécution.

L'instruction INPUT (entrer)

Tapez le programme suivant. (Nous nous dispenserons désormais de représenter le symbole ↵ pour indiquer ENTRÉE à chaque fin de ligne.)

```
10  INPUT A
20  PRINT A;A*2;A*3
30  END
```

Maintenant exécutez ce programme en tapant comme d'habitude la commande RUN. Vous obtenez sur l'écran :

```
?■
```

Le symbole «?» (point d'interrogation) apparaît sur l'écran suivi du curseur pour vous indiquer que le programme attend l'introduction de données.

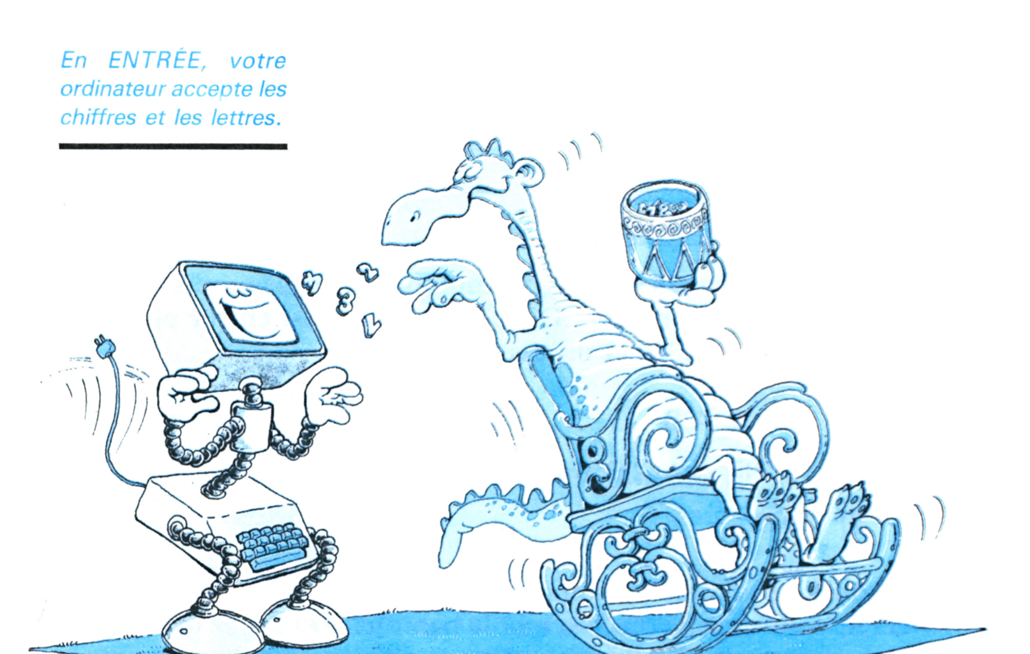
Entrez maintenant un nombre au clavier, par exemple 3, et terminez comme d'habitude en appuyant sur la touche ENTRÉE pour confirmer l'entrée. Vous obtenez sur l'écran :

```
3  6  9
Ready
■
```

Votre programme a été exécuté. Voyons ce qui s'est passé. La première ligne était :

```
10  INPUT A
```

En ENTRÉE, votre ordinateur accepte les chiffres et les lettres.



Cette instruction vous demandait d'entrer un nombre au clavier. Le programme a affiché un ? (point d'interrogation) et s'est arrêté là : il a attendu que vous entriez une valeur. La valeur 3 fournie a ensuite été lue et stockée dans A. A s'appelle une *variable.* Une *variable* est une appellation attribuée à un emplacement mémoire. Voici quelques exemples de variables : A, B, C, F, Z1, G2. Le BASIC de l'Amstrad autorise des noms de variables comportant plusieurs lettres. Par exemple : NOMBRE1, NOM2, SOMME, TAXE.

La seconde instruction était :

```
20  PRINT A;A*2;A*3
```

Sa fonction a été d'imprimer 3, 2 * 3, 3 * 3 ou :

```
3 6 9
```

En utilisant l'instruction INPUT on peut aussi entrer plusieurs variables à la fois. Voici un exemple. Tapez :

```
10 INPUT A,B
20 PRINT A;A*2;B;B*2
30 END
```

Lançons ce programme, comme vous savez maintenant le faire. Sur l'écran vous devez voir le symbole habituel «?». Tapez deux nombres, par exemple 2 et 3, en les séparant par une virgule, puis enfoncez la touche ENTRÉE. A présent vous obtenez sur l'écran :

```
2 4 3 6
```

Examinons ce qui s'est passé. La première ligne du programme était :

```
10 INPUT A,B
```

La fonction de cette instruction a été de vous demander deux valeurs qui ont été stockées par la suite dans les variables A et B. Répétons que les variables sont des noms attribués à des emplacements mémoire. Ici A et B étaient vides à l'origine et on y a mis respectivement 2 et 3. La seconde instruction du programme était :

```
20 PRINT A ;A * 2 ;B ;B * 2
```

Sa fonction a été d'imprimer 2, 2 * 2, 3, 3 * 2, ce qui donne pour résultat :

```
2 4 3 6
```

Lançons à nouveau ce programme. Et cette fois, essayez de taper :

```
5,8
```

Vous obtenez :

```
5 10 8 16
```

Nous pouvons lancer ce programme à plusieurs reprises, entrer au clavier de nouvelles valeurs et obtenir de nouveaux résultats. Il est devenu d'un intérêt plus général puisqu'en nous servant de noms de variables (A et B), au lieu de valeurs définies, nous le rendons apte à être réutilisé autant de fois que nous le souhaiterons.

Pour qu'il soit plus facilement réutilisable nous allons accroître sa lisibilité. Si après l'avoir sauvegardé sur cassette ou sur disquette, nous le laissons quelque temps de côté, nous risquons, lorsque nous le reprendrons, d'avoir oublié ses objectifs. Récrivons-le de façon qu'il affiche sur l'écran des informations plus explicites :

```
10 PRINT"CE PROGRAMME MULTIPLIE CHACUN DES DEUX"
20 PRINT"NOMBRES PAR DEUX* TAPEZ DEUX NOMBRES"
30 INPUT A,B
40 PRINT"PREMIER NOMBRE:";A,"DOUBLE:";2*A
50 PRINT"DEUXIEME NOMBRE:";B,"DOUBLE:";2*B
60 END
```

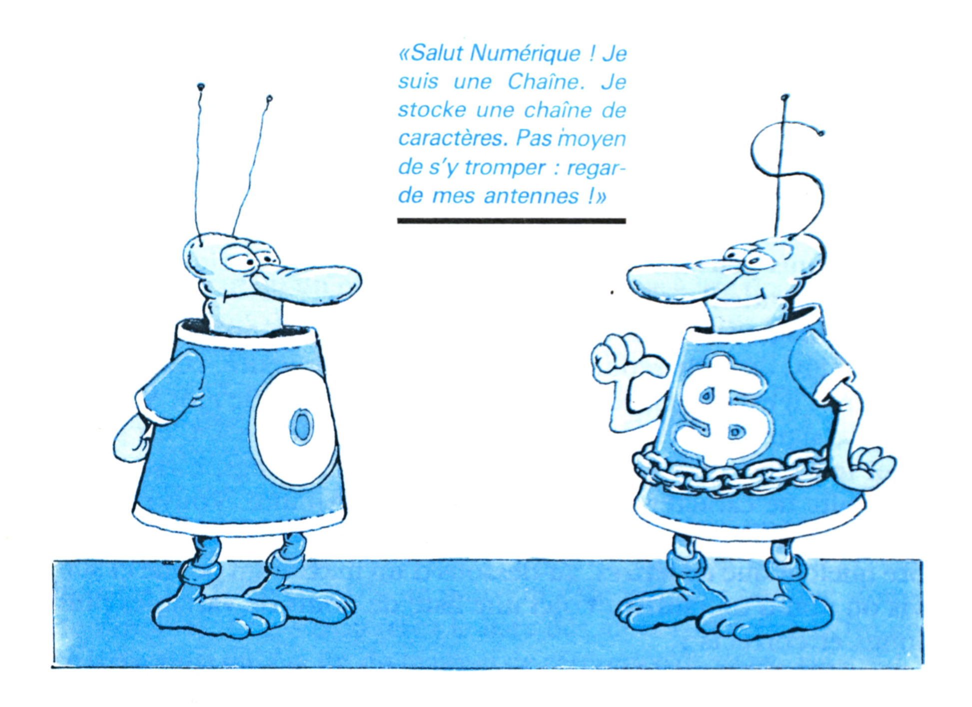
on obtiendra : (Nous allons désormais afficher les données fournies par l'utilisateur en caractères gras.)

```
CE PROGRAMME MULTIPLIE CHACUN DES DEUX
NOMBRES PAR DEUX* TAPEZ DEUX NOMBRES
? 5,7
PREMIER NOMBRE: 5          DOUBLE: 10
DEUXIEME NOMBRE: 7         DOUBLE: 14
```

Nous savons maintenant entrer des données numériques en nous servant de l'instruction INPUT. Nous avons aussi introduit le concept de variable. Voyons comment utiliser efficacement ces techniques et comment mettre au point des programmes plus complexes.

Les deux types de variables

Il existe en BASIC deux *types* de variables : *les variables numériques* et *les variables chaîne de caractères.* Les variables numériques représentent des nombres et les variables chaîne de caractères, du texte. Ce sont deux types différents de variables qui ne s'utilisent pas de la même façon : on peut par exemple additionner des nombres, mais pas du texte. Une variable chaîne de caractères est suivie du symbole «$» (dollar). Voyons tout d'abord les variables numériques.



Variables numériques

Au début de ce chapitre nous nous sommes servi de deux variables numériques appelées A et B. Nous leur avons attribué une valeur en entrant des nombres au clavier. Voyons les règles à respecter pour fixer le nom d'une variable.

Pour nommer une variable, le BASIC de l'Amstrad autorise l'emploi d'une lettre suivie d'une ou de plusieurs lettres ou d'un chiffre. Voici quelques exemples de noms de variables corrects :

A	*(une lettre)*
Z	*(une lettre)*
B1	*(une lettre et un chiffre)*
A2	*(une lettre et un chiffre)*
AB	*(deux lettres)*
AZ	*(deux lettres)*

D'après la définition que l'on vient de donner, les noms suivants sont incorrects :

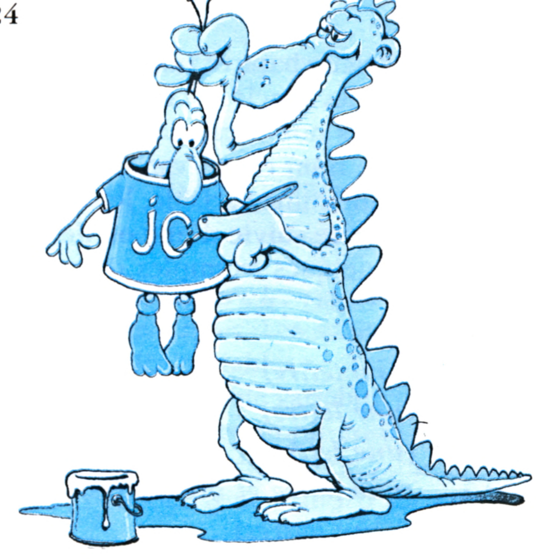
12 *(ne commence pas par une lettre)*
1B *(doit commencer par une lettre)*

Ces noms peuvent être suivis du signe % pour indiquer que la variable contient des valeurs entières.

L'avantage des noms courts est qu'ils n'exigent qu'un interpréteur BASIC de taille réduite et de moindre complexité. Leur inconvénient : il est difficile de les retenir. Par exemple, le mot SOMME est plus significatif et plus facile à retenir que la lettre S. C'est pourquoi le BASIC de l'Amstrad accepte des noms longs, c'est-à-dire une suite de caractères, ce qui améliore la lisibilité. Vous pouvez généralement employer des noms de variables comportant un nombre quelconque de lettres, au-dessous d'un maximum donné et suivis ou non d'un chiffre. Voici une liste de noms de variables plus longs et corrects :

GAGNANT	ETUDIANT1
PERDANT	ETUDIANT2
REGISTRE	ETUDIANT14
SOMME	CASE24

Chaque variable doit avoir un nom.



Les noms suivants sont, eux, incorrects :

3FOIS *(commence par un chiffre)*
UN-1 *(symbole incorrect)*

Il est évident qu'un programme est plus facile à lire quand il comporte des noms de variables longs et explicites. Dans ce chapitre, nous utiliserons des noms longs et des noms courts pour vous habituer aux deux conventions. Rappelons que les noms longs sont utilisés pour une question de commodité et n'affectent en aucune façon le programme.

Il existe une autre restriction dans le choix des noms de variables : on ne peut pas employer un *mot réservé* qui a une signification bien définie pour l'interpréteur BASIC. Par exemple, les commandes LIST, END, RUN ne peuvent être utilisées comme noms de variables. (Vous trouverez la liste des mots réservés à la fin de ce livre ainsi qu'à la fin du manuel de référence de votre Amstrad.) Le BASIC de l'Amstrad interdit même qu'*une partie* du nom de variable soit un mot réservé. Par exemple, OR (OU) est un mot réservé standard et de ce fait PORT ne pourra être utilisé comme nom de variable.

Maintenant que nous savons choisir des noms de variables numériques corrects, apprenons à faire de même avec les variables *chaîne de caractères.*

Variables chaîne de caractères

Voyons d'abord ce que sont les chaînes de caractères. En voici quelques exemples :

```
"RESULTAT"
"VOICI UN EXEMPLE"
"MON NOM EST JOHN"
"25 FOIS 4 ="
```

«Je suis une Variable Chaîne de caractères. Je contiens du texte. Mon nom se termine par $.»



Notons qu'une chaîne de caractères est en principe délimitée par des guillemets afin qu'on ne la confonde pas avec un nom de variable. Une chaîne de caractères peut contenir n'importe quelle suite de caractères à l'exception du symbole qui sert à délimiter la chaîne.

Une chaîne comporte au maximum 256 caractères.

Quand la valeur stockée dans une variable est non pas un nombre mais un texte (c'est-à-dire une chaîne de caractères), la variable s'appelle ***variable chaîne de caractères.***

Le nom d'une variable chaîne de caractères est identique à celui d'une variable numérique, à la différence qu'il doit être suivi du symbole «$» (dollar). Voici quelques noms de variables chaîne de caractères corrects :

A$
BR$
A1$
B5$

Voici quelques exemples corrects de noms longs :

NOM$ (identique à NO$)
VILLE$ (identique à VI$)
UNITE2$ (identique à UN$)

Souvenez-vous qu'un nom tel que GRANDEUR par exemple sera refusé puisqu'il contient le mot réservé AND.

Apprenons à nous servir de variables chaîne de caractères en prenant un programme qui salue l'utilisateur par son nom :

```
10  PRINT"JE SUIS HAL L'ORDINATEUR"
20  PRINT"PRENOM";
30  INPUT PRENOM$
40  PRINT"NOM";
50  INPUT NOM$
60  PRINT"BONJOUR, ";PRENOM$;" ";NOM$;"!"
70  PRINT"JE SAIS MAINTENANT COMMENT TU
    T'APPELLES"
80  PRINT"J'AIME ";PRENOM$;" COMME PRENOM"
90  END
```

Voici le résultat de ce programme. Les caractères que vous entrez apparaissent en caractères gras.

JE SUIS HAL L'ORDINATEUR

PRENOM? **ALBERT**

NOM? **DUPONT**

BONJOUR, ALBERT DUPONT!

JE SAIS MAINTENANT COMMENT TU T'APPELLES

J'AIME ALBERT COMME PRENOM

Vous pouvez désormais communiquer avec votre ordinateur ! Examinons quelques fonctions de base de ce programme. Commençons par la ligne 20.

```
20  PRINT"PRENOM";
```

Notons que cette ligne se termine par un point-virgule qui signifie «afficher le caractère suivant immédiatement après ce texte». Le résultat de l'instruction 20 et votre réponse à l'instruction 30 donnent :

PRENOM? **ALBERT**

Si vous aviez écrit :

```
20  PRINT"PRENOM"
```

sans point-virgule à la fin, le résultat aurait été :

PRENOM

? **ALBERT**

Vous pouvez évidemment opter pour l'une ou l'autre présentation. Ce n'est qu'une affaire de goût. Mais répétons toutefois que si vous voulez que les caractères apparaissent sur une même ligne, il faut un point-virgule à la fin de l'instruction PRINT. Dans le cas contraire, une nouvelle ligne est créée.

Maintenant que nous en savons davantage sur les variables numériques et les variables chaîne de caractères, nous pouvons nous en servir pour continuer notre dialogue avec HAL, l'ordinateur. Ajoutons ce qui suit à notre programme initial.

```
 90  PRINT"EN QUELLE ANNEE SOMMES-NOUS(2CHIFS)";
100  INPUT A1
110  PRINT"EN QUELLE ANNEE ETES-VOUS NE(2 CHIFS)";
120  INPUT A2
130  PRINT"CHER(E) ";PRENOM$;" CETTE ANNEE"
```

```
140  PRINT"TU AS OU TU AURAS";A1-A2;" ANS."
150  END
```

Vous obtiendrez par exemple (ce que vous entrez est en caractères gras) :

```
EN QUELLE ANNEE SOMMES-NOUS(2 CHIFS)? 84
EN QUELLE ANNEE ETES-VOUS NE(2 CHIFS)? 58
CHER(E) ALBERT CETTE ANNEE
TU AS OU TU AURAS 26 ANS.
```

Examinons ce programme en détail. L'instruction :

```
90  PRINT"EN QUELLE ANNEE SOMMES-NOUS(2 CHIFS)";
```

imprime le message entre guillemets. On a de nouveau utilisé le point-virgule pour que les deux chiffres soient affichés sur la même ligne. Dans l'instruction suivante :

```
100  INPUT A1
```

A1 est une variable numérique et la valeur 84 que vous avez tapée sera lue et stockée dans cette variable. Dorénavant, chaque fois que l'identificateur A1 sera utilisé, l'interpréteur BASIC le remplacera automatiquement par la valeur 84. Il en sera de même pour l'instruction 110. L'instruction :

```
110  PRINT"EN QUELLE ANNEE ETES-VOUS NE(2CHIFS)";
```

est du même type que l'instruction 90. Notez le point-virgule à la fin. L'instruction :

```
120  INPUT A2
```

est identique à l'instruction 100. A2 est une nouvelle variable numérique. Elle va bientôt contenir la valeur 58. Dans l'instruction suivante (130), nous utilisons ce nom de variable. Notons que la valeur 58 est automatiquement substituée à la variable. Examinons les instructions 130 et 140 :

```
130  PRINT"CHER(E) ";PRENOM$;" CETTE ANNEE"
140  PRINT"TU AS OU TU AURAS";A1-A2;" ANS."
```

Une fois exécutées, vous voyez sur l'écran :

```
CHER(E) ALBERT CETTE ANNEE
TU AS OU TU AURAS 26 ANS.
```

Décomposons le message en ses éléments :

CHER(E)

(Il s'agit là d'une chaîne de caractères dite littérale, *c'est-à-dire une chaîne de caractères qui fait partie intégrante d'une instruction.)*

ALBERT

(C'est la valeur de la chaîne de caractères entrée au clavier et stockée dans la variable chaîne de caractères appelée PRENOM$. Elle y reste tant que vous n'utilisez pas la commande NEW ou tant que vous n'entrez pas une nouvelle valeur dans PRENOM$.)

CETTE ANNEE TU AS OU TU AURAS

(C'est une autre chaîne de caractères littérale.)

26
(C'est le résultat de A1 — A2, c'est-à-dire 84 — 58 = 26.)

ANS.
(Autre chaîne de caractères littérale.)

Sans doute en tapant et en étudiant ce programme avez-vous déjà éprouvé un sentiment de frustration. Par exemple, vous aviez peut-être envie d'entrer des dates précises (jour et mois) afin de pouvoir calculer votre âge exact à ce jour. Pour cela, il faudrait comparer la date d'aujourd'hui avec celle de votre naissance. Cela n'est possible qu'en faisant appel à une nouvelle instruction BASIC :

IF (condition) THEN (faire...) [Si (condition) Alors (faire...)]

que nous étudierons en détail au Chapitre 7. Peut-être avez-vous envie d'écrire votre programme de telle sorte que l'exécution soit répétitive (ainsi vous n'auriez pas besoin de taper RUN à chaque fois). Nous apprendrons à le faire au Chapitre 6 avec l'instruction GOTO (aller à).

Vous voilà familiarisé avec les variables numériques et les variables chaîne de caractères. Voyons comment les utiliser dans un programme plus long, d'abord en affectant une valeur à une variable, et ensuite en nous servant de la technique du *compteur.*

Affectation d'une valeur à une variable
L'instruction LET (SOIT)

Jusqu'à présent, la seule façon d'affecter une valeur à une variable était d'utiliser l'instruction INPUT. Par exemple, lors de l'exécution de :

```
20 INPUT A
```

vous entrez une valeur telle que 5.2 (suivie par ↵) et la valeur de A est 5.2.

Toutefois, il existe une autre façon d'affecter une valeur à A. C'est : *l'instruction d'affectation.* En voici un exemple :

```
10  A=5.2
```

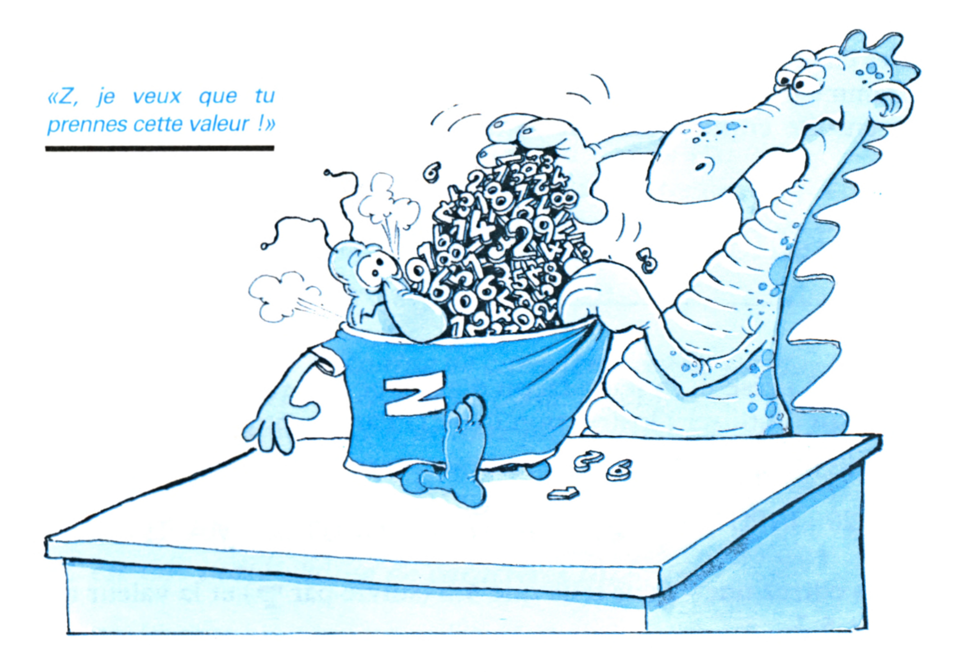
Cette instruction affecte à A la valeur 5.2 à l'intérieur du programme ; (vous n'avez pas besoin de la taper au clavier). Vous pourriez aussi écrire :

```
10  B=1
20  C=2
30  A=B+C
```

Comme vous le constatez, la valeur de A sera établie par 2 + 1 = 3 quand l'instruction 30 sera exécutée.

Examinons maintenant le rôle de l'instruction d'affectation en prenant deux exemples. Dans l'un et l'autre nous programmons le calcul de la somme et la moyenne de deux nombres. Voici le premier programme sans instruction d'affectation :

«Z, je veux que tu prennes cette valeur !»



```
10  PRINT"DONNEZ-MOI 2 NOMBRES JE VAIS CALCULER"
20  PRINT"LEUR SOMME ET LEUR MOYENNE"
30  PRINT"LE PREMIER S.V.P:";
40  INPUT A
50  PRINT"LE SECOND S.V.P:";
60  INPUT B
70  PRINT"LA SOMME DE";A;"ET";B;"EST:";A+B
80  PRINT"LEUR MOYENNE EST:"; (A+B)/2
90  END
```

Nous obtenons :

```
RUN
DONNEZ-MOI 2 NOMBRES JE VAIS CALCULER
LEUR SOMME ET LEUR MOYENNE
LE PREMIER S.V.P:? 24
LE SECOND S.V.P:? 41
LA SOMME DE 24 ET 41 EST: 65
LEUR MOYENNE EST: 32.5
```

Notons que l'expression A + B est répétée deux fois dans les instructions PRINT. Ce n'est là qu'un inconvénient mineur. Cependant, dans un programme plus long, le risque d'erreurs est plus grand. Et si vous devez modifier le programme, pour utiliser une formule différente par exemple, la réécriture sera plus longue.

Voici donc un programme équivalent qui utilise une *variable intermédiaire* appelée SOMME permettant de stocker le résultat. La lisibilité est meilleure et surtout le risque d'erreurs est moindre.

```
10  PRINT"DONNEZ-MOI DES NOMBRES JE VAIS CALCULER"
20  PRINT"LEUR SOMME ET LEUR MOYENNE"
30  PRINT"LE PREMIER S.V.P:";
40  INPUT A
50  PRINT"LE SECOND S.V.P:";
60  INPUT B
80  SOMME=A+B
90  MOYENNE=SOMME/2
100 PRINT"LEUR SOMME EST:";SOMME
110 PRINT"LEUR MOYENNE EST:";MOYENNE
120 END
```

Nous avons utilisé deux nouvelles variables :

```
80  SOMME=A+B
90  MOYENNE=SOMME/2
```

L'utilisation de noms de variables supplémentaires présente deux avantages : le programme est plus clair et plus facile à modifier. Supposons par exemple que nous voulions maintenant modifier ce programme pour obtenir la moyenne de trois nombres. Pour cela, nous devons taper :

```
72  PRINT"LE TROISIEME S.V.P:";
75  INPUT C
80  SOMME=A+B+C
90  MOYENNE=SOMME/3
```

La suite du programme reste inchangée. Nous avons simplement entré un troisième nombre et modifié les formules à un seul endroit. Voici le programme complet :

```
10 PRINT"DONNEZ-MOI DES NOMBRES JE VAIS CALCULER"
20 PRINT"LEUR SOMME ET LEUR MOYENNE"
30 PRINT"LE PREMIER S.V.P:";
40 INPUT A
50 PRINT"LE SECOND S.V.P:";
60 INPUT B
72 PRINT"LE TROISIEME S.V.P:";
75 INPUT C
80 SOMME=A+B+C
90 MOYENNE=SOMME/3
100 PRINT"LEUR SOMME EST:";SOMME
110 PRINT"LEUR MOYENNE EST:";MOYENNE
120 END
```

Nous obtenons :

```
DONNEZ-MOI DES NOMBRES JE VAIS CALCULER
LEUR SOMME ET LEUR MOYENNE
LE PREMIER S.V.P:? 5
LE SECOND S.V.P:? 3
LE TROISIEME S.V.P:? 10
LEUR SOMME EST: 18
LEUR MOYENNE EST: 6
```

Nous venons de voir deux méthodes d'affectation d'une valeur à une variable. Nous pouvons utiliser, soit l'instruction INPUT : une valeur est entrée lors de l'exécution du programme, soit l'instruction d'affectation : une valeur ou une méthode de calcul de la valeur (une formule) est stockée dans le programme. On choisira :

— la première méthode (avec l'instruction INPUT) si l'on sait que la valeur explicitement fournie à la variable (c'est-à-dire une valeur *non calculée*) sera différente à chaque exécution du programme.
— la deuxième méthode (avec l'instruction d'affectation) chaque fois que l'on emploiera une formule pour calculer la valeur de la variable ou si l'on sait que sa valeur explicite (c'est-à-dire une valeur *non calculée*) sera la même à chaque exécution du programme.

Voyons maintenant toutes les règles d'écriture d'une instruction d'affectation.

La syntaxe d'une affectation

La syntaxe (ou ensemble des règles qui régit l'écriture d'une instruction d'affectation) est simple. Celle-ci se présente habituellement sous la forme :

< variable > = < expression >

Il doit toujours y avoir une variable à gauche et une expression à droite.
Plus précisément, une expression est :

— un nombre ou une variable, ou

— un nombre ou une variable suivie d'un opérateur (tel que +, —, *, /) et d'une autre expression.

Voici quelques exemples :

3 *(nombre)*
A *(variable)*
2 + 2 *(nombre, opérateur, nombre)*
A + B * 3 *(variable, opérateur, expression)*

Les expressions peuvent être insérées entre parenthèses, par exemple :

3 + (A + 2)/2

B + ((C * 2) + (D/2)/4)

Vous pouvez considérer une expression comme une valeur, ou comme quelque chose qui sera calculé et dont le résultat sera une valeur (en d'autres termes, une formule de calcul d'une valeur).

Le signe égal (=) utilisé dans l'instruction d'affectation n'a pas la même signification que celle qu'il a en mathématiques. Dans une expression il signifie «Prend la valeur de». Vous pouvez, par exemple, écrire :

```
10  A = 1
20  A = A + 1
```

(En mathématiques l'expression A = A + 1 est absurde.) En BASIC, une fois l'instruction 20 exécutée, la valeur de A sera 1 (la valeur initiale de A) + 1 = 2. Pour éviter toute confusion, actuellement certains langages de programmation utilisent le symbole ← ou := au lieu de = . Répétons qu'en BASIC le signe = signifie que la variable située à gauche prend la valeur de l'expression située à droite dans une instruction d'affectation.

Voici quelques exemples d'instructions d'affectation correctes :

A = — 3 + 2 *(— 3 est un entier négatif)*

B = A + 1

C = (2 * 3) + (A/B)

```
MOYENNE = SOMME/NOMBRE
CARRE = A ↑ 2
X = B ↑ 2 — (4 * A * C)
```

Examinons cette dernière instruction d'affectation et voyons si elle répond à notre définition :

```
B ↑ 2
```

correspond à :

< variable > < opérateur > < nombre >

suivie de :

```
— (4 * A * C)
```

c'est-à-dire :

< opérateur > < expression entre parenthèses comportant une valeur >

Entre les parenthèses :

```
4 * A * C
```

correspond à :

< nombre > < opérateur > < variable > < opérateur > < variable >

C'est donc là une expression correcte.
Au contraire les instructions suivantes sont incorrectes :

B + C = SOMME *(seule une variable (et non une expression) peut être placée à gauche du signe =)*

2 = A	*(à gauche il doit y avoir une variable et non une valeur)*
SOMME = B + C (D/3)	*(après C il manque un opérateur)* *(Cette instruction est acceptée par l'Amstrad mais donne un résultat erroné.)*
(MOYENNE) = (B + C)/2	*(parenthèses interdites à gauche)*
A =	*(il manque une valeur à droite)*

Enfin, notons qu'au moment où une instruction d'affectation est exécutée, toutes les variables situées à droite du signe = doivent prendre une valeur. Si vous écrivez :

```
10  B=2
20  SOMME=B+C
30  INPUT C
```

le programme ne donnera pas le résultat désiré car à la ligne 20 la valeur de C n'a pas encore été définie. Sans doute vouliez-vous écrire :

```
10  B=2
20  INPUT C
30  SOMME=B+C
```

Nous connaissons maintenant la syntaxe des affectations. Nous allons nous en servir pour introduire une technique importante utilisant les affectations : la technique du compteur. Nous y aurons souvent recours dans nos programmes.

La technique de la variable compteur

Répétons qu'une variable est simplement un nom donné à un emplacement mémoire. L'instruction INPUT ou l'instruction d'af-

fectation (=) permet de stocker une valeur dans la mémoire. Dans notre exemple, nous voulons modifier la valeur d'une variable de façon répétitive pour pouvoir compter des événements. La technique utilisée est celle de la *variable compteur.*

Voyons comment une suite d'affectations peut changer la valeur de la variable N. Tapez ce qui suit en mode immédiat (ce mode est aussi appelé *mode calculateur*) :

```
N = 1
```

La valeur est automatiquement stockée dans N. Vérifions-le en tapant :

```
PRINT N
```

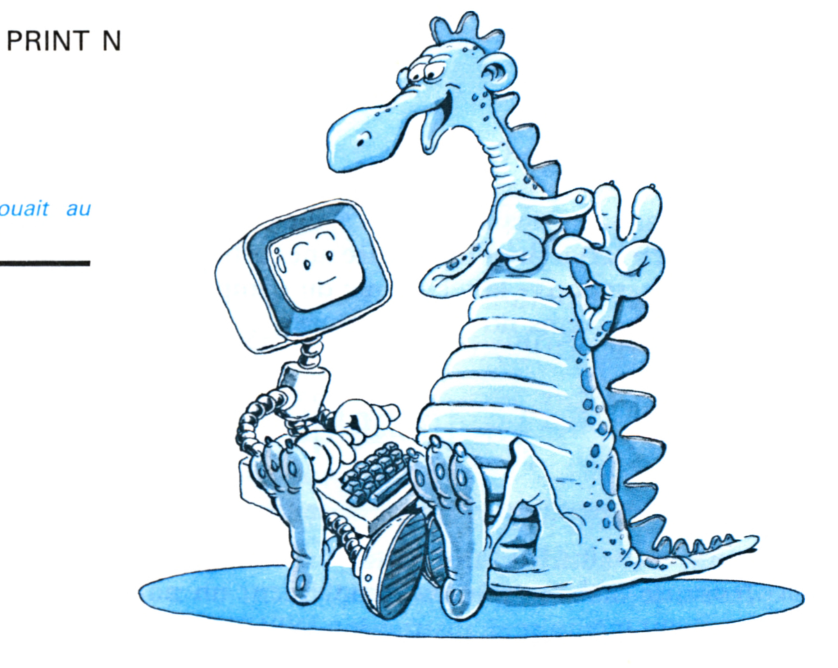
La valeur 1 apparaît. Maintenant tapez :

```
N = 2
```

N contient donc 2. Tapez :

```
PRINT N
```

«Et si on jouait au compteur ?»



La réponse est :

```
2
```

La valeur 1 a été remplacée par la valeur 2 dans N. Tapez :

```
N = 3
```

puis :

```
PRINT N
```

et vérifiez que la valeur 3 a bien été stockée dans N. Le mécanisme fonctionne. Nous allons bientôt utiliser cette technique pour compter des événements. En d'autres termes, une variable peut servir de compteur d'événements. Voici un aperçu d'un programme plus compliqué qui comptabilise les entrées de nombres au clavier. Il s'arrête quand vous frappez le zéro.



«Je suis une variable compteur.»

```
10  SOMME=0
20  SOMME=SOMME+1
30  PRINT"TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER"
40  INPUT NOMBRE
50  PRINT"VOUS AVEZ TAPE";SOMME;"NOMBRE(S)"
60  IF NOMBRE < > 0 THEN 20
70  END
```

Nous examinerons plus en détail des instructions telles que l'instruction 60 dans le Chapitre 6. Cette instruction signifie : si le NOMBRE n'est pas égal (< >) à zéro, ALORS exécuter l'instruction 20. Nous obtenons :

```
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 5
VOUS AVEZ TAPE 1 NOMBRE(S)
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 1
VOUS AVEZ TAPE 2 NOMBRE(S)
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 2
VOUS AVEZ TAPE 3 NOMBRE(S)
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 3
VOUS AVEZ TAPE 4 NOMBRE(S)
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 4
VOUS AVEZ TAPE 5 NOMBRE(S)
TAPEZ UN NOMBRE. TAPEZ 0 POUR ARRETER
? 0
VOUS AVEZ TAPE 6 NOMBRE(S)

READY.
```

Dans ce programme, la valeur de la SOMME est *initialisée* à 0 dans la première instruction. Chaque fois que vous entrez un nombre, la somme augmente de 1. C'est une variable compteur. (Les programmes proposés dans la suite de cet ouvrage vont nous fournir l'occasion d'appliquer cette technique.) Plus tard, nous apprendrons à maîtriser un tel programme afin que, lorsque nous l'arrêtons, 0 ne compte pas pour un nombre.

Résumé

Dans ce chapitre, nous avons appris à écrire des programmes qui pourraient être continuellement réutilisés et dont les résultats seront

fonction des valeurs entrées au clavier. Pour cela nous avons utilisé des variables.

Une variable est le nom donné à un emplacement mémoire dans lequel on peut stocker une valeur numérique ou un texte.

Nous avons appris à changer le contenu d'une variable au moyen de l'instruction INPUT et de l'instruction d'affectation. Nous pouvons désormais écrire un petit programme réalisant un dialogue ou des calculs simples.

Remarquons toutefois que nos programmes comportent maintenant plus de dix lignes. Il est indispensable qu'ils soient d'une parfaite lisibilité. Nous allons donc apprendre à écrire clairement un programme.

Exercices

4-1 : Entrez 4 nombres au clavier et affichez le total, la moyenne et le produit de ces quatre nombres.

4-2 : Les identificateurs suivants sont-ils corrects ?

a - 24B	g - INPUT
b - B24	h - INPUT1
c - A+B	i - PI
d - APLUSB	j - 3$
e - ALPHA2D	k - TROIS
f - EXEMPLE	l - NOM$

4-3 : Ecrivez un programme qui demande à l'utilisateur de dire son nom et qui répond :

«JE PENSE QUE JE CONNAIS UN (NOM) !»

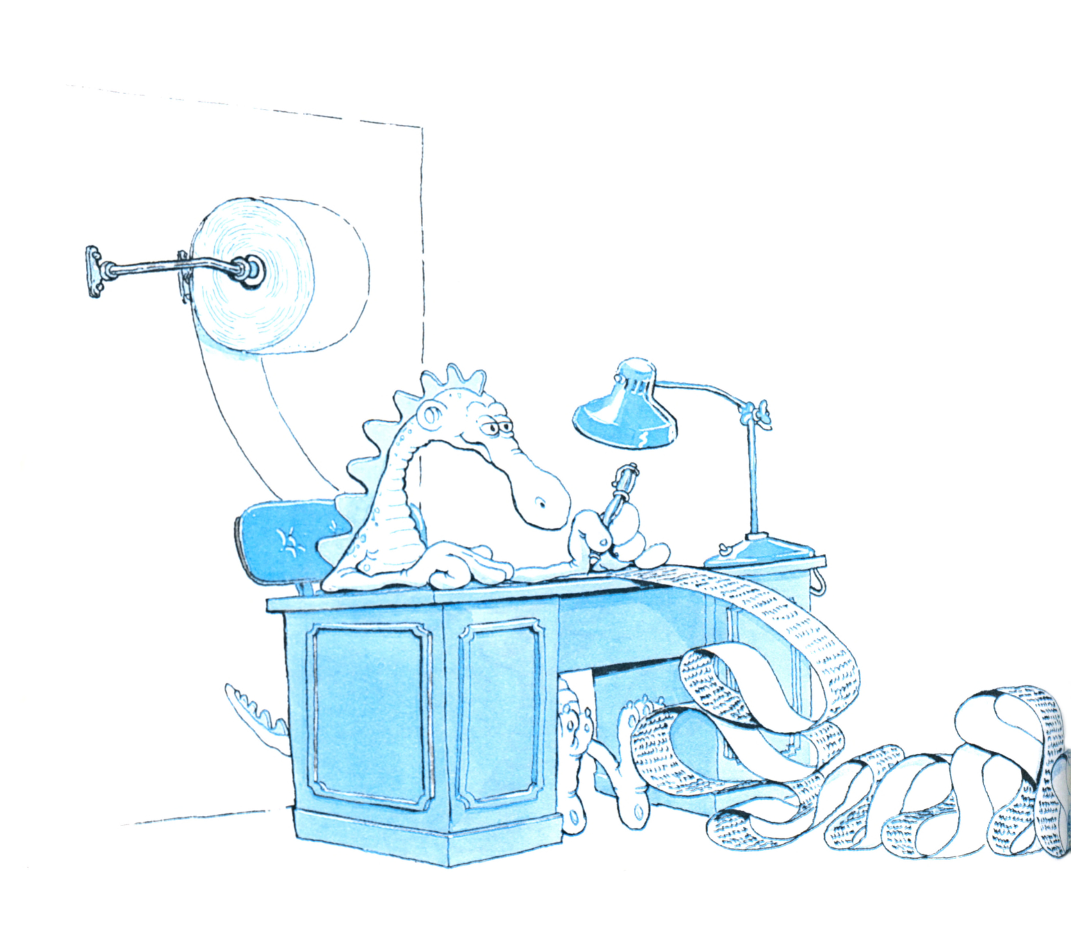
4-4 : Ecrivez un programme qui demande :
- le nom d'un objet
- le nom d'un meuble
- le nom d'un ami

et répond : «EST-CE QUE VOTRE AMI (NOM) A UN(E) (OBJET) SUR UN(E) (MEUBLE)»

4-5 : Les instructions ci-dessous sont-elles correctes ?

a - A + 1 = A
b - A = A + A + A
c - A = B + C
d - B + C = A
e - 3 = 2 + 1
f - VALEUR = PREMIER + SECOND * 2



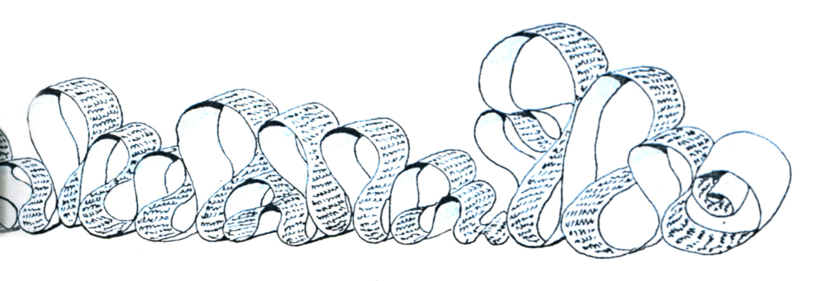


5

ECRIRE UN PROGRAMME CLAIR

Jusqu'à présent, nous avons écrit de petits programmes comportant trois types différents d'instructions : PRINT, INPUT et l'affectation (=). Nous avons aussi appris à écrire des expressions arithmétiques simples. Dans les prochains chapitres, nous verrons de nouveaux types d'instructions qui vont nous permettre d'écrire des programmes plus longs. Mais avant de continuer, apprenons à écrire des programmes clairs et lisibles. C'est de la plus grande importance. Nous étudierons dans ce but sept techniques :

1. L'utilisation de l'instruction REM (c'est-à-dire l'introduction de REMarques dans un programme).
2. L'utilisation de plusieurs instructions sur une même ligne.
3. L'utilisation de blancs à l'intérieur d'une instruction.
4. L'utilisation de l'instruction «PRINT vide» et de l'instruction CLS (pour améliorer l'affichage à l'écran).
5. L'utilisation de l'instruction «INPUT condensé» (pour réduire le nombre de lignes dans un programme).
6. Le choix de noms de variables significatifs.
7. La numérotation de ligne adéquate.

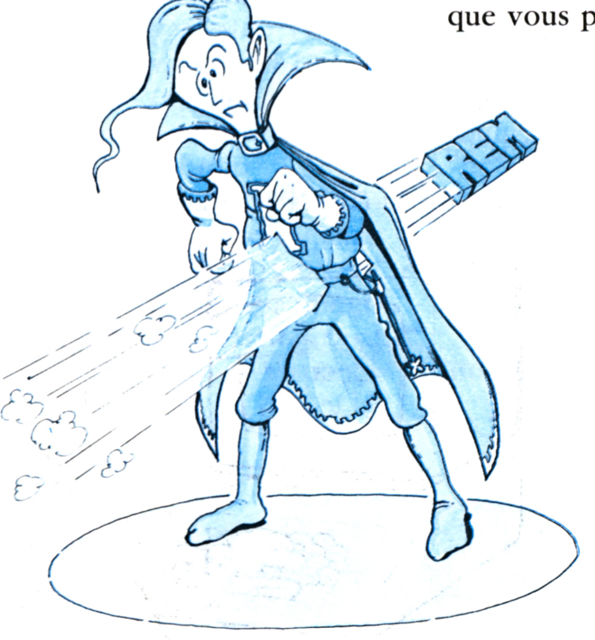


L'instruction REM

Voici un exemple d'emploi de l'instruction REM :

```
10 REM ***PROGRAMME ADDITION***
20 PRINT"DONNEZ-MOI 2 NOMBRES:";
30 INPUT PRE,SEC
40 PRINT"LEUR SOMME EST:";PRE+SEC
50 END
```

En fait on a recours à l'instruction REM pour rendre le programme plus lisible en introduisant des commentaires dans le texte. Lors de l'exécution du programme, l'interpréteur ignore cette instruction et elle n'a aucun effet sur le programme. Voici quelques exemples d'instructions que vous pouvez utiliser :



L'interpréteur ne voit pas les REMarques.

```
25   REM LIRE LES DEUX NOMBRES
45   REM NOUS POURRIONS AUSSI LES MULTIPLIER DANS
     L'INSTRUCTION CI-DESSUS
```

Voici ce que l'on obtient :

```
10   REM ** ADDITION **
20   PRINT"DONNEZ-MOI 2 NOMBRES:";
25   REM LIRE LES DEUX NOMBRES
30   INPUT PRE,SEC
40   PRINT"LEUR SOMME EST:";PRE+SEC
45   NOUS POURRIONS AUSSI LES MULTIPLIER DANS
     L'INSTRUCTION CI-DESSUS
50   END
```

Pour améliorer encore la lisibilité, n'hésitez pas à employer des étoiles, des tirets ou autres symboles.

```
10   REM *** ADDITION ***
60   REM — SECONDE PARTIE —
100  REM ==GRAND FINAL==
200  REM $$$$$ CHANGER CETTE PARTIE PLUS TARD $$$$$
```



«Regardez comme ce programme est plus lisible quand on lui ajoute quelques étoiles !»

Instructions multiples sur une même ligne

Le BASIC de l'Amstrad autorise l'écriture de deux ou plusieurs instructions sur la même ligne, en général séparées par deux points. En voici un exemple :

```
50 PRINT"TAPEZ UN NOMBRE";:INPUT NOMBRE
```

est équivalent à :

```
50 PRINT"TAPEZ UN NOMBRE";
60 INPUT NOMBRE
```

Notons qu'il ne peut y avoir qu'une étiquette par ligne. De ce fait, si vous écrivez deux instructions sur une même ligne, il n'y aura qu'une étiquette sur la gauche. Voici un autre exemple :

```
100 REM **FAIRE TOUS LES CALCULS SUR UNE LIGNE **
110 SOMME=A+B:PRODUIT=A*B:MOYENNE=SOMME/2
```

La ligne 110 comporte trois instructions.

Cette méthode est efficace dans deux cas au moins : pour clarifier une instruction INPUT et pour introduire une REMarque à droite de l'instruction. En voici un exemple :

```
70 PRINT"TAPEZ 2 NOMBRES";:INPUT N1,N2
```

La ligne 70 est écrite pour correspondre à ce qui se passe sur l'écran, ce qui rend le programme plus lisible. Un autre exemple : on a inséré une REMarque sur la même ligne que l'instruction à laquelle elle se réfère.

```
60 RESULTAT=A+2*B—5:REM LE RESULTAT DOIT ETRE
POSITIF
```

Autre exemple :

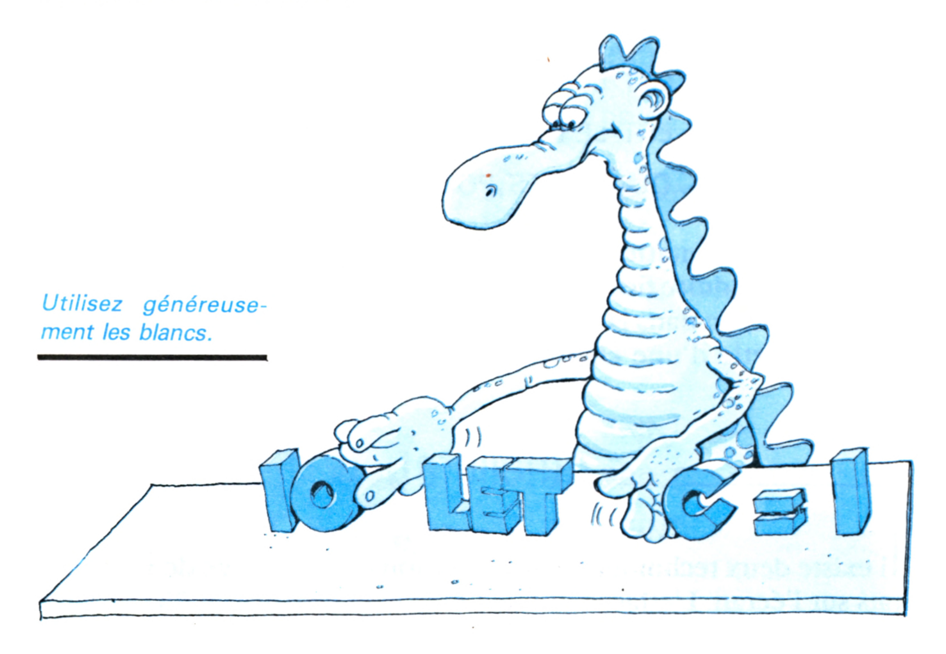
```
50  TEMPERATURE=(FAHR—32)*5/9:REM CONVERSION EN
    CELSIUS
```

Utilisation de blancs (ou espaces)

Le BASIC ignore généralement les blancs sauf dans les noms, les chaînes de caractères ou les entrées de données. Par exemple, vous pouvez écrire :

```
20  PRINT 4+2*3
```

Il est toutefois difficile de lire une telle instruction. En utilisant largement les blancs, on facilite la relecture du programme. Vous pouvez utiliser des blancs :



Utilisez généreusement les blancs.

— *après chaque mot réservé, tel que PRINT et INPUT :*

```
50  PRINT 4
60  INPUT NOMBRE
```

— *avant et après chaque opérateur :*

```
30  PRINT 4 + 2 * 3
40  RESULTAT = A1/((B - C) * D)
```

Enfin voici un exemple d'utilisation des blancs à l'intérieur d'une instruction REM :

```
1  REM PROGRAMME D'INVENTAIRE
2  REM COPYRIGHT 1983
3  REM LES VARIABLES
4  REM C   EST LA COULEUR (1 A 10)
5  REM U   EST LE NOMBRE D'UNITES (JUSQU'A 1000)
6  REM T   EST LA TAILLE (1 A 50)
7  REM CT EST LE COUT UNITAIRE
8  REM PX EST LE PRIX UNITAIRE
9  REM Q   QUANTITES POUR REASSORTIMENT
```

Une telle présentation facilite la lecture. Toutefois, vous ne pouvez pas introduire de blancs au milieu d'un mot réservé, d'un nom de variable, ou dans la réponse à un INPUT, à moins que les blancs fassent partie d'une chaîne de caractères.

Amélioration de l'affichage

Il existe deux techniques d'amélioration de l'affichage de vos résultats sur l'écran. L'effacement de l'écran et l'instruction PRINT vide.



La simplicité avant tout !

L'instruction CLS permet d'effacer l'écran. De ce fait, vous pouvez commencer chaque nouveau programme par l'instruction :

```
10  CLS
```

Voici une astuce à utiliser à la fin de votre programme :

```
100  CLS:LIST
```

Cette instruction efface votre écran et vous donne la liste des instructions du programme.

L'instruction PRINT vide s'utilise pour afficher une ligne blanche en tapant simplement :

```
50  PRINT
```

Si vous désirez sauter trois lignes, tapez :

```
50  PRINT:PRINT:PRINT
```

ou encore :

```
50  PRINT
60  PRINT
70  PRINT
```

INPUT condensé

Chaque fois que vous utilisez une instruction INPUT, vous devez expliquer à l'utilisateur du programme ce qu'il faut entrer. Voici un exemple :

```
50  PRINT"TAPEZ 2 NOMBRES";
60  INPUT N1,N2
```

Le BASIC de l'Amstrad autorise l'instruction INPUT condensé ; vous pouvez donc écrire :

```
50  INPUT"TAPEZ 2 NOMBRES";N1,N2
```

L'instruction est équivalente à celles mentionnées ci-dessus. Cette technique réduit la frappe et montre clairement ce qui sera affiché sur l'écran.

Choix des noms de variables

Attribuez toujours à chaque variable un nom qui vous permette de l'identifier immédiatement. Cela vous évitera des erreurs quand vous devrez écrire un long programme. De plus, si vos noms de variables ne sont pas clairs, il pourrait vous arriver, quand vous reprendrez votre travail après un certain délai, de ne pas retrouver aisément les objectifs de votre programme.

Votre BASIC autorise des noms à plusieurs lettres, utilisez-les. Voici quelques exemples :

```
10  REM VOICI LES VARIABLES
20  REM  RE    POUR RESULT
30  REM  N1    POUR NOMBRE 1
40  REM  N2    POUR NOMBRE 2
50  END
```

Cependant, il faut tenir compte d'une restriction importante à savoir que :

— vous ne pouvez pas utiliser de mots réservés tels que PRINT, REM, INPUT comme noms de variables ni à l'intérieur des noms de variables.

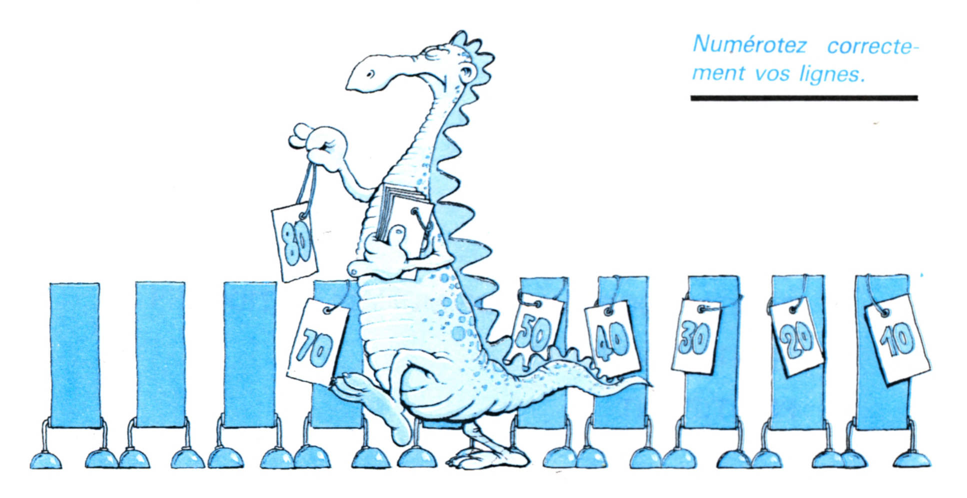
Il est préférable d'identifier tous les noms de variables ainsi que les formules ou équations au début de chaque programme.

Numérotation de ligne correcte

Jusqu'à maintenant, nous avons numéroté les lignes par multiple de 10 :

```
10   (instruction)
20   (instruction)
30   (instruction)
```

Numérotez correctement vos lignes.



Toutefois, vous pouvez choisir n'importe quelle série d'étiquettes, à condition de n'utiliser que des entiers positifs et de ne pas aller au-delà des limites imposées par l'interpréteur. Par exemple, vous pouvez écrire :

```
1     (instruction)
2     (instruction)
3     (instruction)
```

ou

```
100   (instruction)
200   (instruction)
300   (instruction)
```

Nous avons pris la précaution de laisser un intervalle régulier entre les étiquettes successives pour que vous puissiez facilement par la suite faire des corrections, ou ajouter des instructions. Par exemple, voici la version 1 d'un programme :

```
10  REM ** PROGRAMME MULTIPLICATION**
20  INPUT"DONNEZ-MOI 2 NOMBRES";N1,N2
30  PRINT"LE PRODUIT EST"; N1 * N2
40  END
```

Nous voulons maintenant clarifier ce programme et améliorer l'affichage. Pour cela nous taperons des instructions supplémentaires, à savoir :

```
 5  CLS
15  PRINT"PROGRAMME DE MULTIPLICATION"
17  PRINT:PRINT:PRINT
35  PRINT:PRINT
```

Ces nouvelles instructions seront automatiquement insérées dans le programme. Tapons la commande LIST et voyons ce qui apparaît sur l'écran :

```
LIST

5  CLS
10 REM **PROGRAMME MULTIPLICATION**
15 PRINT"PROGRAMME DE MULTIPLICATION"
17 PRINT:PRINT:PRINT
20 INPUT"DONNEZ-MOI 2 NOMBRES";N1,N2
30 PRINT"LE PRODUIT EST:";N1*N2
35 PRINT:PRINT
40 END
Ready
■
```

Nous obtiendrons par exemple :

```
PROGRAMME DE MULTIPLICATION
DONNEZ-MOI 2 NOMBRES? 12,15
LE PRODUIT EST: 180
```

Si vous prévoyez de faire de nombreuses corrections ou d'ajouter plusieurs instructions, vous pouvez laisser de plus grands espaces dans la numérotation des lignes et utiliser par exemple :

10 (instruction)

50 (instruction)

60 (instruction)

100 (instruction)

Résumé

Ecrire des programmes qui fonctionnent suppose avant tout de la discipline. Il est important d'être aussi ordonné et aussi bien organisé que possible. L'utilisation d'instructions condensées augmente le risque d'erreurs. N'hésitez pas à prendre le temps de rendre plus clairs vos programmes et leur affichage. Dans ce chapitre, nous avons décrit les techniques permettant d'écrire des programmes clairs.

Dès à présent, efforcez-vous d'acquérir de bonnes habitudes de programmation. Suivez les conseils et suggestions qui vous sont donnés ici. C'est la condition nécessaire pour que vos programmes, qui vont devenir de plus en plus complexes, soient lisibles et fonctionnent sans problème.



Exercices

5-1 : Décrivez les techniques qui améliorent la lisibilité de l'affichage.

5-2 : Les instructions ci-dessous sont-elles correctes ?

a. A = A + 1
b. A = 3 + 1
c. PRINT ALPHA + 2
d. SOMME = 2 + (3 + (4/5))/2
e. IN PUT NOMBRE
f. SOMME = 2 2 + 3 3

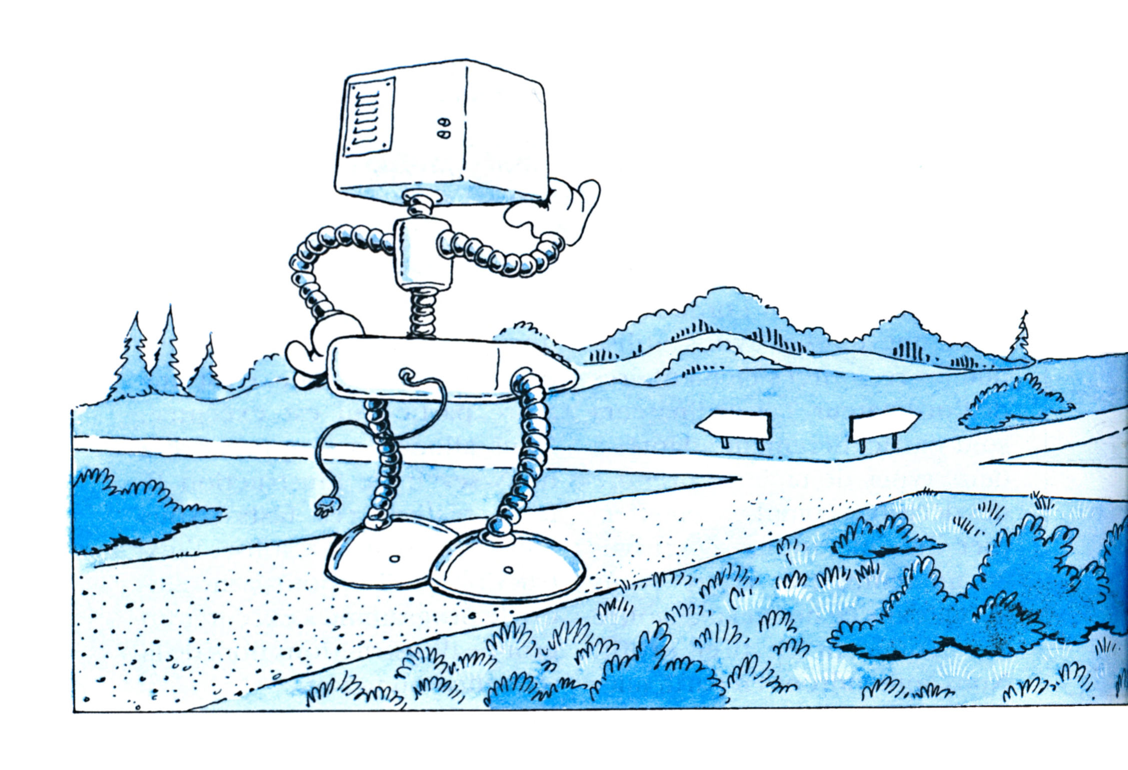
5-3 : Donnez la raison pour laquelle la plupart des INPUT doivent être précédés d'un message.

5-4 : Ecrivez trois exemples d'INPUT condensés.

5-5 : Pourquoi utilise-t-on des REMs ?

5-6 : Quelle est la valeur de A d'après ces deux instructions :

```
30 A = 3
40 REM A = 4
```



6

PRENDRE DES DECISIONS

Jusqu'à présent, nous avons appris à communiquer avec l'ordinateur et à exécuter des calculs arithmétiques simples. Toutefois, nos programmes ne sont guère intéressants et nous pourrions exécuter les mêmes tâches sans faire intervenir l'ordinateur. C'est que nous n'avons utilisé que les ressources élémentaires de l'ordinateur. Les ordinateurs sont particulièrement adaptés à deux types de tâches : prendre des décisions complexes et effectuer des opérations répétitives en un temps très court. Nous allons dans ce nouveau chapitre voir comment en tirer profit. Nous apprendrons en particulier à écrire des programmes prenant des décisions. Nos programmes vont devenir «intelligents» puisqu'ils vont décider de ce qu'ils doivent faire.

Dans les programmes BASIC, les décisions sont prises en testant une valeur au moyen de l'instruction IF. Si le test réussit, une partie du programme est exécutée. S'il échoue, c'est une autre partie qui est exécutée. Nous allons donc apprendre à nous servir de l'instruction IF pour réaliser ces tests. Nous apprendrons aussi à utiliser l'instruction GOTO pour contraindre le programme à exécuter un groupe d'instructions hors de la séquence numérique.

L'instruction IF s'écrit :

IF (condition) THEN (instruction, c'est-à-dire faire quelque chose) ce qui se traduit par SI (condition) ALORS (instruction)

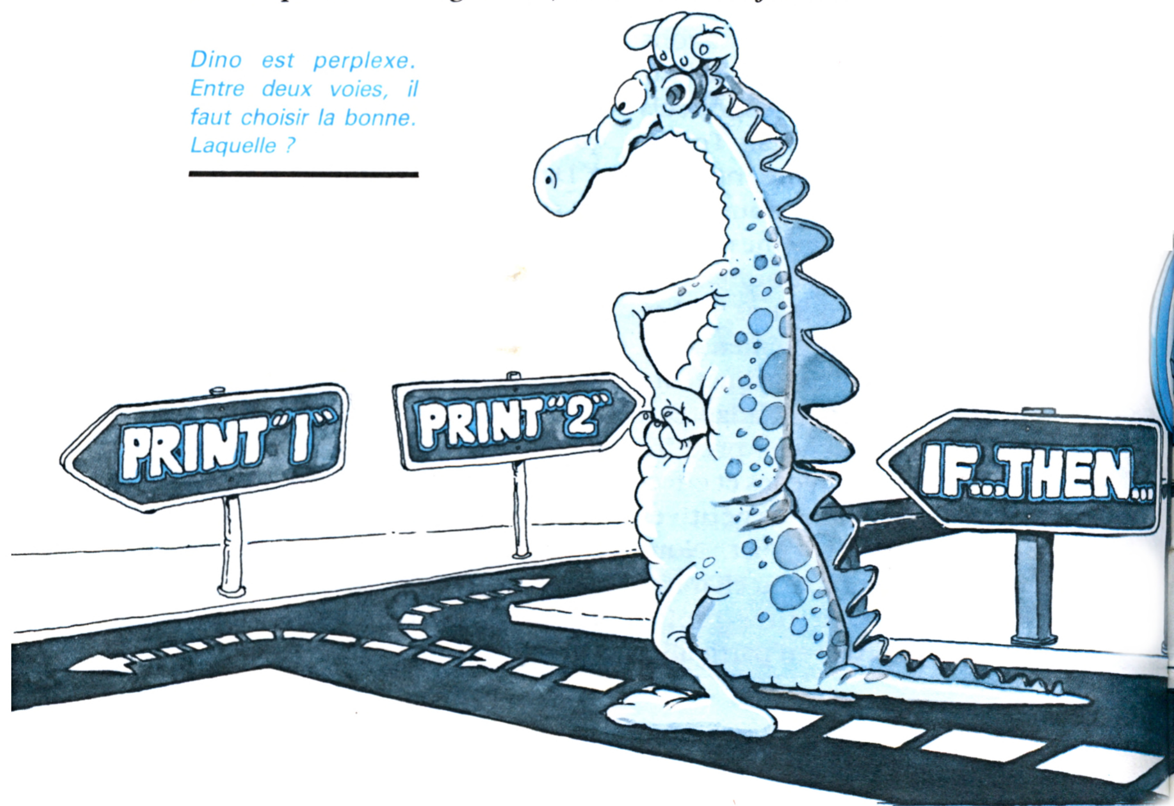
En voici un exemple :

```
IF I=1 THEN PRINT"UN"
```

La fonction d'une telle instruction est claire : SI (IF) la valeur de la variable I est égale à 1 au moment où cette instruction est exécutée, ALORS (THEN) le mot UN est imprimé. SI I est différent de 1, il ne se passe rien et l'instruction suivante du programme est exécutée.

I = 1 est appelé une *expression logique*. L'expression I = 1 est *vraie* quand I est égal à 1 ; sinon elle est *fausse*.

Dino est perplexe. Entre deux voies, il faut choisir la bonne. Laquelle ?



L'instruction IF...THEN (SI...ALORS) vous permet de tester la valeur d'une expression et d'exécuter une instruction ou une autre - c'est-à-dire prendre une décision - selon les résultats du test. Voici un autre exemple :

```
10  INPUT I
20  IF I=1 THEN PRINT"UN"
30  END
```

Maintenant lancez ce programme. Tapez «1» au clavier. Sur l'écran, vous voyez :

```
RUN
? 1
UN
```

Relancez ce programme et tapez «2». Sur l'écran, vous voyez maintenant :

```
RUN
? 2
```

Cette fois-ci, il n'y a pas de message imprimé en réponse à 2.

Apprenons à notre programme à reconnaître les nombres de 1 à 4 :

```
10  REM CE PROGRAMME RECONNAIT LES NOMBRES DE
    1 A 4
20  INPUT"TAPEZ UN ENTIER:";NOMBRE
30  IF NOMBRE=1 THEN PRINT"UN"
```



```
40  IF NOMBRE=2 THEN PRINT"DEUX"
50  IF NOMBRE=3 THEN PRINT"TROIS"
60  IF NOMBRE=4 THEN PRINT"QUATRE"
70  END
```

Lançons ce programme. En voici deux passages types tels qu'ils apparaissent sur l'écran (y compris les caractères gras) :

```
RUN
TAPEZ UN ENTIER:? 3
TROIS
RUN
TAPEZ UN ENTIER:? 5
```

C'est correct mais pas encore parfait. L'idéal serait que, lorsque nous tapons 5, le programme donne par exemple comme réponse :

```
JE NE CONNAIS PAS CE NOMBRE
```

ou bien qu'il vous demande un nouveau nombre entier.

Il y a une fonction caractéristique de l'instruction IF qui donne ce type de réponse. Par exemple, vous pouvez écrire :

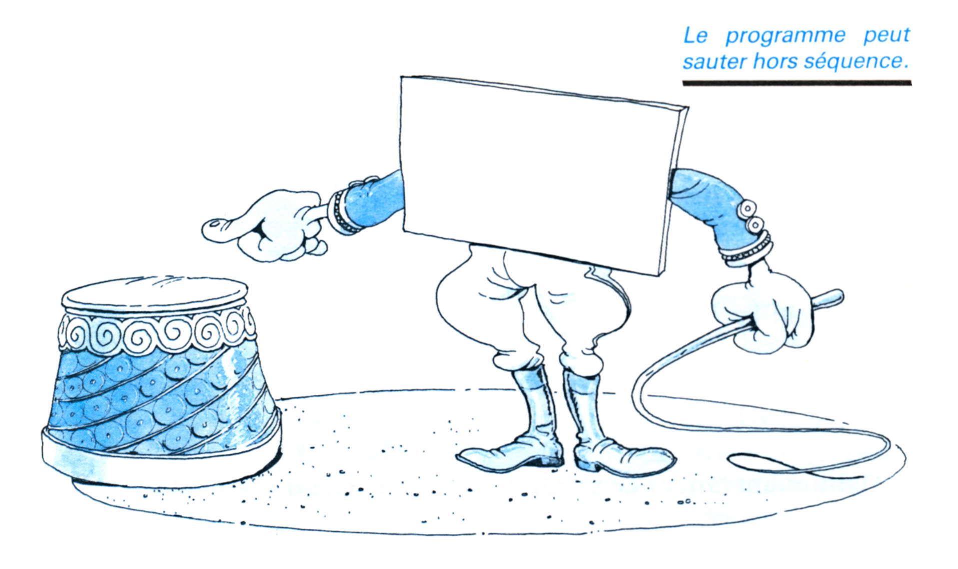
```
70  IF NOMBRE=5 THEN 20
```

où 20 correspond au numéro de la ligne à exécuter si le test réussit. Il s'agit là d'une nouvelle forme de l'instruction IF signifiant que, si le NOMBRE est égal à 5, il faut alors passer à la ligne 20. Nous savons donc maintenant sortir d'une séquence. En voici un exemple :

```
10  INPUT I
20  IF I=1 THEN 50
30  PRINT"CE N'EST PAS UN 1"
40  END
50  PRINT"C'EST UN 1"
60  END
```

Lancez ce programme et entrez un «1» au clavier. Vous obtenez sur l'écran :

```
RUN
? 1
C'EST UN 1
```



Le programme peut sauter hors séquence.

Lancez à nouveau ce programme et tapez un «2». Cette fois-ci, vous obtenez :

```
RUN
? 2
CE N'EST PAS UN 1
```



Moi, le Bug, je vous ai bien eu !

Notre programme est devenu «intelligent», c'est-à-dire que, dans l'un et l'autre des cas, il envoie le message approprié. Peut-être vous demandez-vous si l'on obtient le même résultat en utilisant la forme d'origine de l'instruction IF. Essayons :

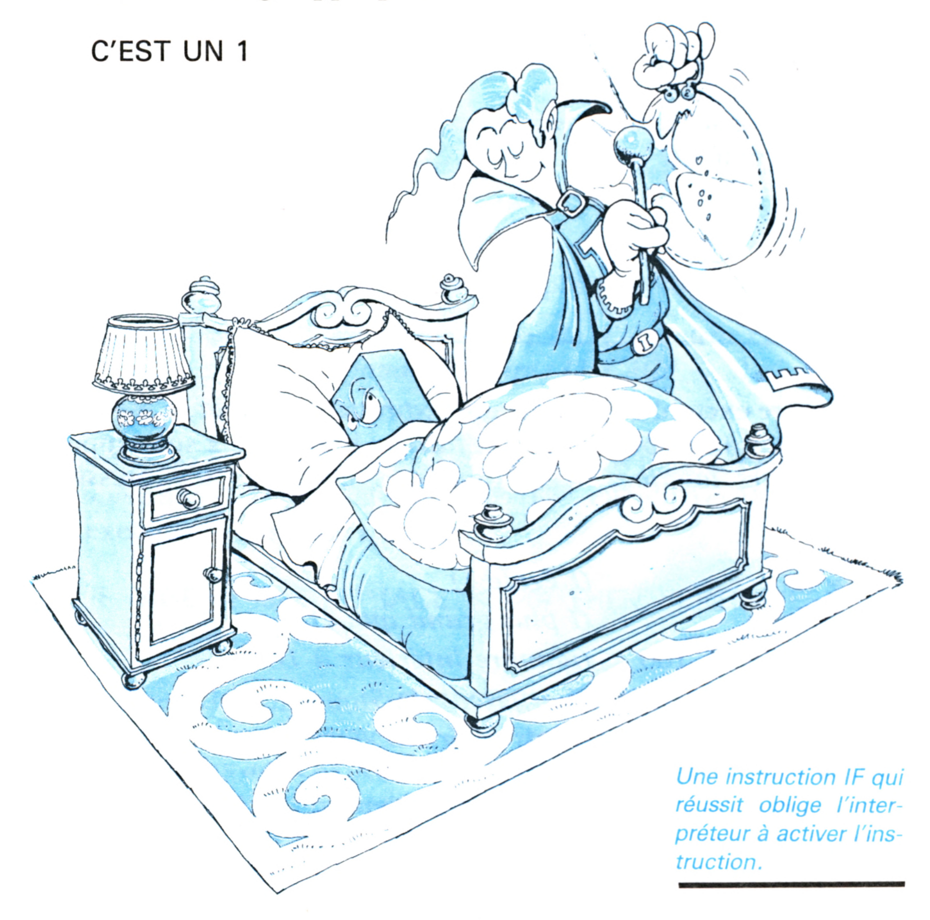
```
10  INPUT I
20  IF I=1 THEN PRINT"C'EST UN 1"
30  PRINT"CE N'EST PAS UN 1"
40  END
```

Maintenant lancez ce programme et entrez un «1» au clavier. Sur l'écran, vous obtenez :

```
RUN
? 1
C'EST UN 1
CE N'EST PAS UN 1
```

Le programme ne fonctionne pas. Sans tenir compte du succès ou de l'échec du test, l'instruction suivante (ici 30) est exécutée. Dans cet exemple, une fois l'instruction IF exécutée, nous obtenons d'abord le message approprié :

```
C'EST UN 1
```



Une instruction IF qui réussit oblige l'interpréteur à activer l'instruction.

Mais ensuite, le second message est lui aussi imprimé :

```
CE N'EST PAS UN 1
```

La nouvelle forme de l'instruction IF

```
IF I = 1 THEN 50
```

élimine ce problème. Nous utiliserons donc souvent cette instruction dans nos programmes.

Etudions de plus près l'instruction IF afin d'inventorier ses possibilités. La forme générale de l'instruction IF (SI)... THEN (ALORS) est :

IF (expression logique) THEN (instruction exécutable ou étiquette)

Examinons successivement les expressions logiques et les instructions exécutables.

Expressions logiques

Dans notre exemple, I = 1 est une *expression logique,* c'est-à-dire qu'elle peut être soit *vraie,* soit *fausse.* Vrai et faux sont appelés des *valeurs logiques.* Voici quelques exemples d'expressions logiques :

I = 1	*(I égale 1)*
I > 4	*(I plus grand que 4)*
NOMBRE < 100	*(nombre inférieur à 100)*
I < > 5	*(I différent de 5)*
MOIS < 13	*(mois inférieur à 13)*

Une expression logique combine des valeurs ou variables et des opérateurs logiques. Pour compléter, voici la liste des opérateurs que vous pouvez utiliser dans des expressions logiques :

= égal
< > différent *(en mathématiques, les symboles sont ≠ ou #)*
< inférieur
> supérieur
< = inférieur ou égal *(en mathématiques, le symbole est ≤)*
> = supérieur ou égal *(en mathématiques, le symbole est ≥)*

Voici quelques expressions logiques complexes :

(NOMBRE + 2) > 4

(AGE — 5) > = 10

((2 * I — 5)/2) < 10

2 > I

Vous pouvez aussi écrire :

4 > 2 *(c'est toujours vrai)*

4 = 2 *(c'est toujours faux)*

L'expression suivante n'est pas une expression logique correcte :

(2 AGE — 2) < 5 *(expression incorrecte : l'opérateur * fait défaut. Il faudrait écrire : (2 * AGE — 2) < 5)*

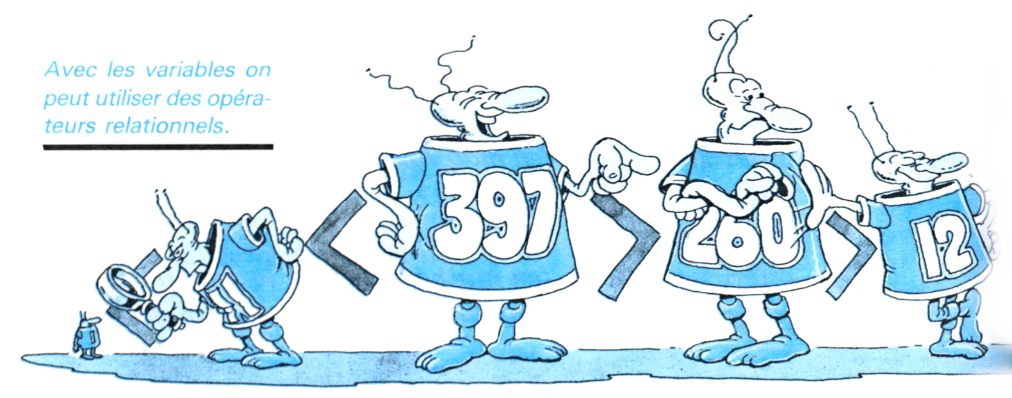
Vous pouvez même combiner des expressions logiques en utilisant des *opérateurs logiques* tels que AND (ET), OR (OU) et NOT (NON). Par exemple, vous pouvez écrire :

```
IF (AGE > 9) AND (AGE < 20) THEN PRINT"C'EST UN
    ADOLESCENT"
```

Cette instruction imprime «C'EST UN ADOLESCENT» chaque fois que l'âge est compris entre 9 et 20. L'expression utilisant AND est vraie, quand les *deux* expressions logiques sont vraies. Notons que vous *ne pouvez pas* écrire :

```
IF AGE ( >9 AND <20) THEN...
```

Avec les variables on peut utiliser des opérateurs relationnels.



étant donné que chaque expression logique doit être insérée entre parenthèses.

Voici un autre exemple utilisant deux expressions logiques :

```
20  INPUT REP$
30  IF (REP$ = ''OUI'') OR (REP$ = ''NON'') THEN 60
40  PRINT''REPONSE INCORRECTE''
50  END
60  PRINT''REPONSE CORRECTE-CONTINUONS''
```

Cette partie de programme fait entrer une réponse dans la variable REP$. (Répétons que le symbole $ situé à la fin du nom signifie que la variable contient une chaîne de caractères.) OUI et NON sont les seules réponses possibles. Ce programme vérifie la validité de ce que vous entrez au clavier. Si vous tapez OUI, alors l'expression (REP$ = ''OUI'') est vraie, et l'instruction IF réussit : l'instruction 60 est ensuite exécutée, et le programme affiche :

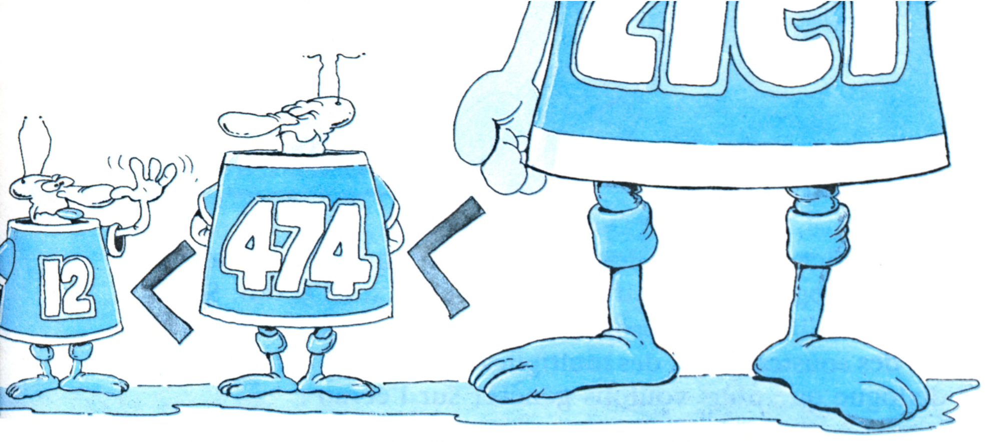
```
REPONSE CORRECTE - CONTINUONS
```

Si vous tapez NON, vous obtenez la même réponse. Si vous tapez n'importe quoi d'autre, vous obtenez :

```
REPONSE INCORRECTE
```

et le programme s'arrête à la ligne 50 (instruction END = FIN).

Une expression utilisant OR (OU) est *vraie*, si au moins un des



deux éléments est vrai. Elle est fausse quand les deux éléments sont faux.

Par exemple, si vous tapez «SI», alors l'expression (REP$ = ''OUI'') est fausse. La seconde expression est alors testée (REP$ = ''NON''). Elle est fausse et le programme affiche le message :

```
REPONSE INCORRECTE
```

Enfin, l'opérateur NOT (NON) peut être utilisé pour nier une condition. Voici un exemple d'instruction IF complexe :

```
IF ((MOYENNE < 3.5) AND (DERNIEREXAMEN < 3.0) AND
    NOT (ORAL > 4.0)) THEN PRINT "ECHEC"
```

Dans cette instruction, nous vérifions trois conditions à la fois. Nous ne poursuivrons pas davantage l'étude d'expressions aussi complexes, qui sortent du cadre de cet ouvrage.

Instructions exécutables

Rappelons la définition de l'instruction IF :

IF (expression logique) THEN (instruction exécutable ou étiquette)

Maintenant que nous sommes familiarisés avec les expressions logiques, examinons la deuxième partie de cette définition :

THEN (instruction exécutable ou étiquette)

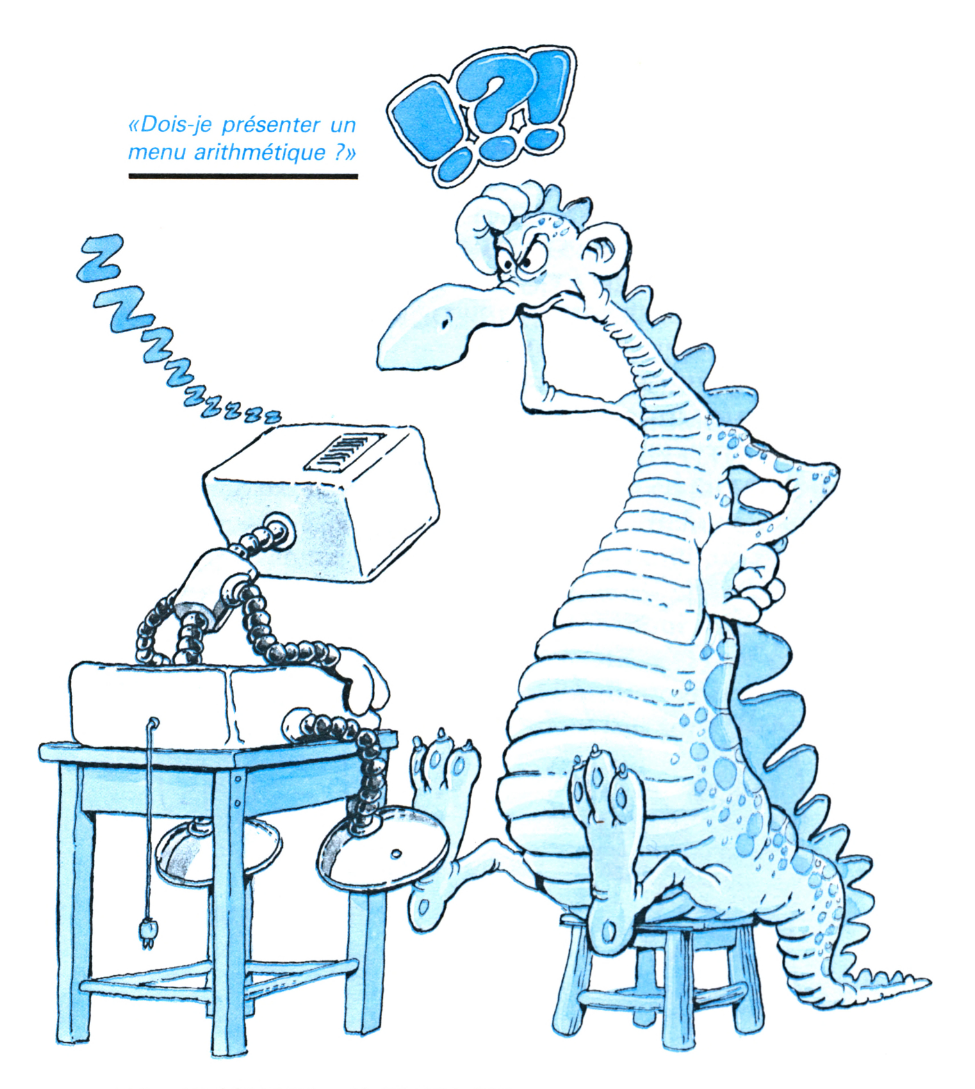
Une instruction exécutable est tout simplement une instruction qui peut être exécutée. Ce peut être une instruction d'assignation, une instruction INPUT ou une instruction PRINT.

Pour vérifier nos nouvelles acquisitions, nous allons mettre au point un programme qui affiche un «menu» sur l'écran. Selon le choix de l'utilisateur, ce programme éducatif va exécuter des additions, des soustractions, des multiplications ou des divisions. Voici le dialogue que nous voulons générer sur l'écran :

```
BIENVENUE AU PROFESSEUR ORDINATEUR.
JE VAIS VERIFIER VOTRE NIVEAU
EN ARITHMETIQUE
QUE VOULEZ-VOUS FAIRE?
— ADDITION                (1)
— SOUSTRACTION            (2)
— MULTIPLICATION          (3)
— DIVISION                (4)
VOTRE CHOIX:? 3
MULTIPLIONS
COMBIEN FONT 2 FOIS 3? 6
FELICITATIONS!!
```

Voici maintenant le programme :

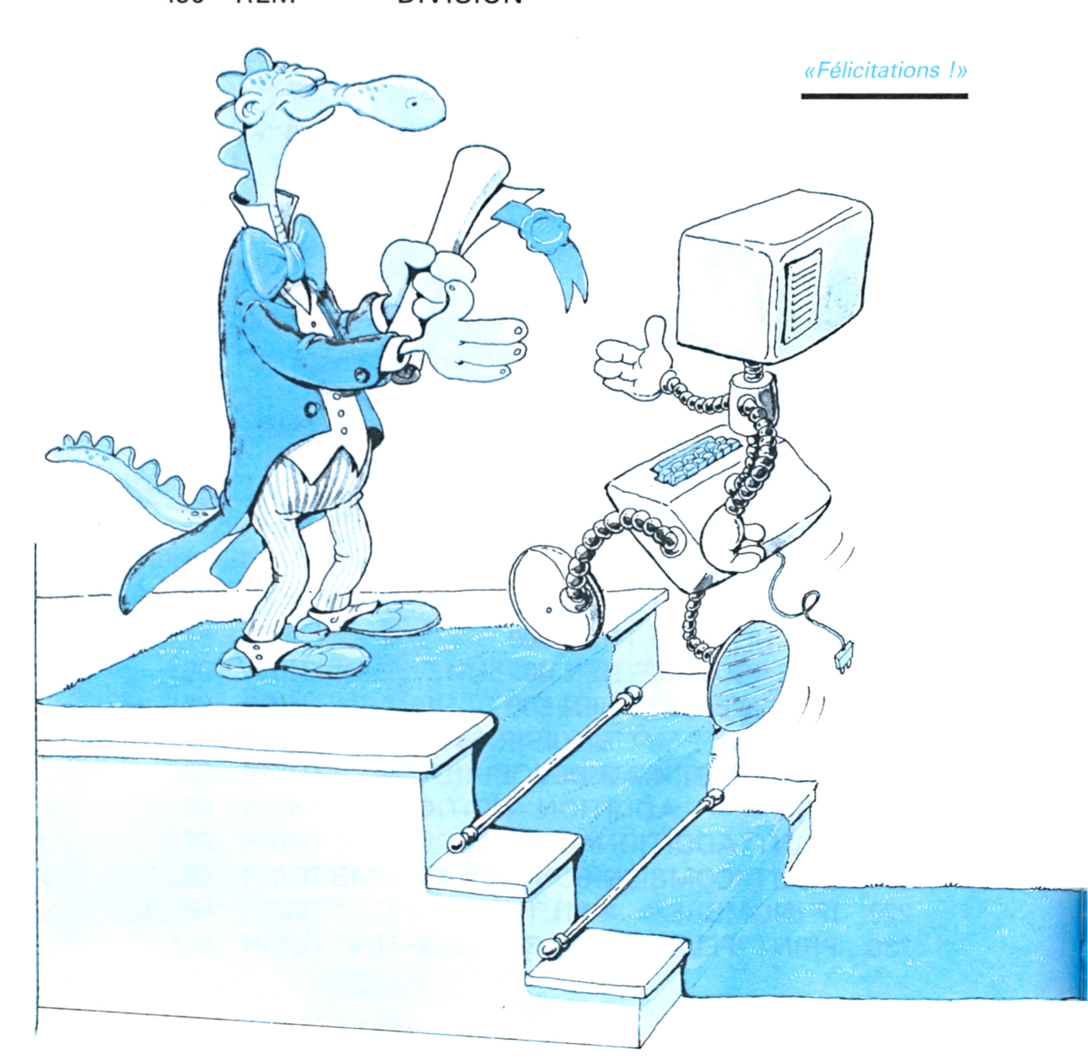
```
10  REM *ARITHMETIQUE*
15  CLS
20  PRINT"BIENVENUE AU PROFESSEUR ORDINATEUR."
30  PRINT"JE VAIS VERIFIER VOTRE NIVEAU"
40  PRINT"EN ARITHMETIQUE"
50  PRINT"QUE VOULEZ-VOUS FAIRE?"
60  PRINT"—ADDITION                   (1)"
70  PRINT"—SOUSTRACTION               (2)"
80  PRINT"—MULTIPLICATION             (3)"
90  PRINT"—DIVISION                   (4)"
100 INPUT"VOTRE CHOIX:"; CHOIX
```



«Dois-je présenter un menu arithmétique ?»

```
110  IF (CHOIX=1) THEN 200
120  IF (CHOIX=2) THEN 300
130  IF (CHOIX=3) THEN 400
140  IF (CHOIX=4) THEN 500
150  PRINT"IL FAUT CHOISIR UN NOMBRE ENTRE"
160  PRINT"1 ET 4.AU REVOIR !!!":END
170  REM ———ADDITION———
200  PRINT"ADDITIONNONS"
210  INPUT"COMBIEN FONT 4+7";NOMBRE
220  IF (NOMBRE<  >11) THEN 600
230  PRINT"FELICITATIONS!!":END
```

```
290  REM ---SOUSTRACTION---
300  PRINT"SOUSTRAYONS"
310  INPUT"COMBIEN FONT 9-5";NOMBRE
320  IF (NOMBRE<  >4) THEN 600
330  PRINT"FELICITATIONS!!":END
390  REM ---MULTIPLICATION---
400  PRINT"MULTIPLIONS"
410  INPUT"COMBIEN FONT 2 FOIS 3";NOMBRE
420  IF (NOMBRE<  >6) THEN 600
430  PRINT"FELICITATIONS!!":END
490  REM ---DIVISION---
```



«Félicitations !»

```
500  PRINT"DIVISONS"
510  INPUT"COMBIEN FONT 9 DIVISE PAR 3";NOMBRE
520  IF (NOMBRE<  >3) THEN 600
530  PRINT"FELICITATIONS!!":END
590  REM ---SORTIE ECHEC---
600  PRINT"FAUX DESOLE!AU REVOIR!":END
```

Ce programme paraît long et imposant, mais en fait il est très simple. Examinons-le. Les instructions 20 à 90 affichent le «menu» sur l'écran. Le programme contrôle le choix de l'utilisateur à l'aide des instructions 110 à 140 (les parenthèses après chaque instruction IF ne sont pas obligatoires ; on les emploie pour augmenter la lisibilité). Si l'utilisateur tape «1» alors (CHOIX = 1) est vrai et l'instruction 200 est exécutée à la suite. Si l'utilisateur tape autre chose que 1, 2, 3 ou 4, les instructions 150 et 160 sont exécutées et le programme affiche :

```
IL FAUT CHOISIR UN NOMBRE ENTRE
1 ET 4.AU REVOIR !!!
Ready
■
```

et s'arrête (instruction END à la ligne 160).

Tapons par exemple 3. L'instruction 110 échoue, et l'instruction 120 est exécutée. L'instruction 120 échoue et l'instruction 130 est exécutée.

L'instruction 130 réussit puisque (CHOIX = 3) est vrai et l'instruction

400 est exécutée. Voici la partie du programme correspondante :

```
400  PRINT"MULTIPLIONS"
410  INPUT"COMBIEN FONT 2 FOIS 3";NOMBRE
420  IF (NOMBRE< >6) THEN 600
430  PRINT"FELICITATIONS!!":END
```

Dans notre exemple, nous tapons 6 en réponse à l'instruction 410. Quand l'instruction 420 est exécutée, l'instruction (NOMBRE < > 6) est fausse puisque NOMBRE = 6. (Répétons que < > signifie «différent de».) L'instruction à exécuter est la ligne 430, et le programme répond :

```
FELICITATIONS!!
Ready
■
```

et s'arrête puisque la ligne 430 contient *deux* instructions. La seconde est :

```
END
```

Supposons maintenant qu'une fois le menu affiché vous tapiez un nombre autre que 1, 2, 3 ou 4 : le programme s'arrête. L'idéal, bien sûr, serait qu'il fasse savoir à l'utilisateur que seuls les nombres compris entre 1 et 4 sont acceptés et lui offre la possibilité de revenir au début. D'une façon générale, il serait satisfaisant de pouvoir reprendre un programme en tel ou tel point de son parcours. Cette solution idéale vous est précisément offerte par l'instruction GOTO.

L'instruction GOTO (aller à)

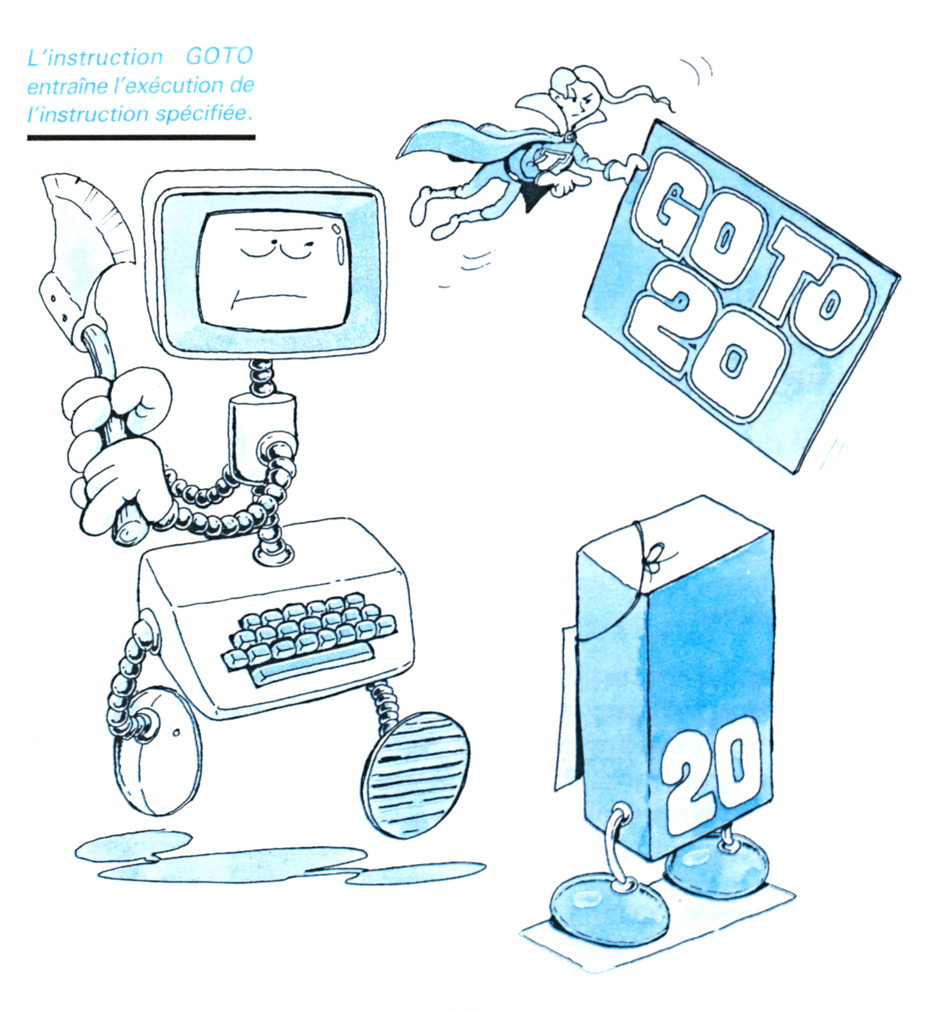
L'instruction GOTO s'écrit :

GOTO (étiquette)

L'instruction définie par cette étiquette est alors exécutée. En voici un exemple :

```
10  PRINT"CE PROGRAMME RECONNAIT LES 1."
20  PRINT"TAPEZ 0 POUR L'ARRETER"
30  INPUT"TAPEZ UN NOMBRE:";NOMBRE
40  IF NOMBRE=1 THEN PRINT"UN"
50  IF NOMBRE=0 THEN 70
60  GOTO 30
70  END
```

L'instruction GOTO entraîne l'exécution de l'instruction spécifiée.



Vous obtenez :

```
RUN
CE PROGRAMME RECONNAIT LES 1.
TAPEZ 0 POUR L'ARRETER
TAPEZ UN NOMBRE:? 1
UN
TAPEZ UN NOMBRE:? 5
TAPEZ UN NOMBRE:? 25
TAPEZ UN NOMBRE:? 1
UN
TAPEZ UN NOMBRE:? 0

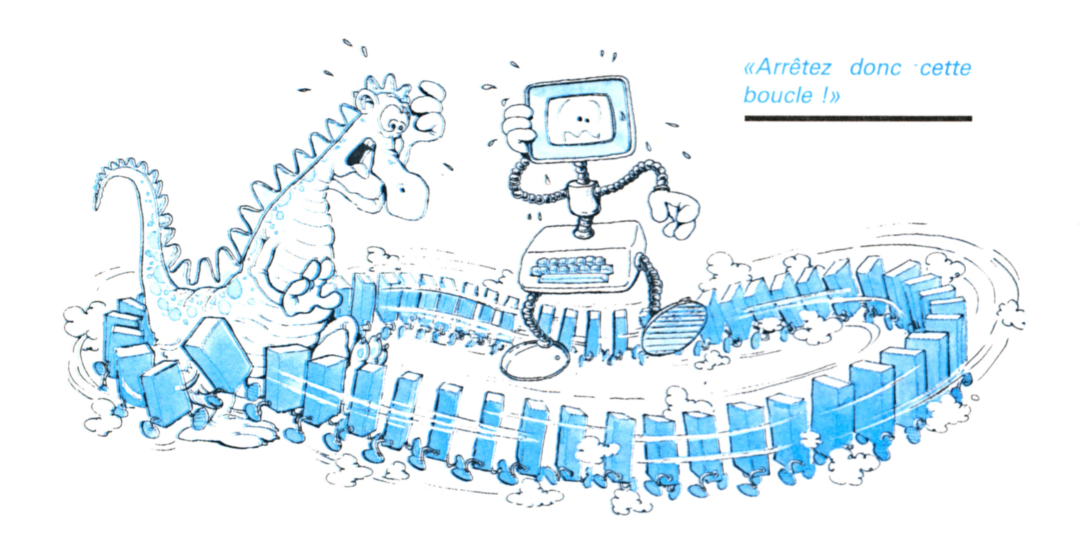
Ready
■
```

Chaque fois que vous tapez 1 le programme le reconnaît et affiche UN. Si vous tapez autre chose, le nombre est ignoré et le programme demande une nouvelle valeur. Le programme revient continuellement au début. C'est une *boucle.* On dit que le programme boucle sur lui-même. Si vous tapez un 0 (zéro), il est détecté par l'instruction 50 et le programme saute à l'instruction de fin 70.

Supprimons maintenant l'instruction 50. Nous obtenons le programme suivant :

```
10  PRINT"CE PROGRAMME RECONNAIT LES 1."
20  PRINT"TAPEZ 0 POUR L'ARRETER"
30  INPUT"TAPEZ UN NOMBRE:";NOMBRE
40  IF NOMBRE=1 THEN PRINT"UN"
50  GOTO 30
60  END
```

Vous pouvez obtenir :



«Arrêtez donc cette boucle !»

```
RUN
CE PROGRAMME RECONNAIT LES 1.
TAPEZ 0 POUR L'ARRETER
TAPEZ UN NOMBRE:? 1
UN
TAPEZ UN NOMBRE:? 5
TAPEZ UN NOMBRE:? 25
TAPEZ UN NOMBRE:? 0
TAPEZ UN NOMBRE:?
```

Tel l'apprenti sorcier, nous avons créé là un terrible problème : ce programme ne va jamais s'arrêter. Il s'agit là d'une erreur de programmation courante appelée *boucle infinie*. Ne vous inquiétez pas, c'est sans risque pour le matériel. Pour arrêter votre programme, tapez la touche ESC 2 fois. Nous allons nous efforcer d'éviter cet inconvénient en fournissant désormais une sortie normale (programmée) à chaque programme.

Nous venons d'introduire l'instruction GOTO ; revenons à notre définition de l'instruction IF et simplifions-là.

L'instruction IF révisée

Rappelons que l'instruction IF peut se présenter sous la forme :

IF (expression logique) THEN (étiquette)

ou sous la forme :

IF (expression logique) THEN (instruction exécutable)

Voici un exemple de la première définition :

```
IF NOMBRE = 0 THEN 60
```

Equivalent à :

```
IF NOMBRE = 0 THEN GOTO 60
```

GOTO 60 est une instruction exécutable, et vous constaterez que la forme :

```
THEN 60
```

est tout simplement une forme abrégée de

```
THEN GOTO 60
```

La forme IF (expression logique) THEN (étiquette) est plus simple que la forme :

IF (expression logique) THEN (instruction exécutable)

Voici une autre simplification : l'utilisation de THEN devant GOTO est facultative. Les formes suivantes sont toutes équivalentes :

```
IF NOMBRE = 0 THEN 100
IF NOMBRE = 0 THEN GOTO 100
IF NOMBRE = 0 GOTO 100
```

Nous allons maintenant utiliser des programmes dans lesquels figurent les instructions IF et GOTO.

Compteur de 1

Au Chapitre 5, nous avons étudié la technique du compteur. Utilisons-la pour compter le nombre de 1 tapés dans le programme employant l'instruction GOTO. Voici le programme révisé :

```
  1  REM COMPTEUR DE 1
 10  PRINT"JE VAIS COMPTER LE NOMBRE DE 1 TAPES"
 20  PRINT"TAPEZ 0 POUR ARRETER"
 30  SOMME=0
 40  INPUT"TAPEZ UN NOMBRE";NOMBRE
 50  IF NOMBRE=0 THEN 100
 60  IF NOMBRE< >1 THEN GOTO 40
 70  SOMME=SOMME+1
 80  PRINT"NOMBRE DE 1 COMPTES: ";SOMME
 90  GOTO 40
100  END
```

Nous obtiendrons par exemple :

```
RUN
JE VAIS COMPTER LE NOMBRE DE 1 TAPES
TAPEZ 0 POUR ARRETER
TAPEZ UN NOMBRE? 10
TAPEZ UN NOMBRE? 1
NOMBRE DE 1 COMPTES: 1
TAPEZ UN NOMBRE? 9
TAPEZ UN NOMBRE? 5
TAPEZ UN NOMBRE? 1
NOMBRE DE 1 COMPTES: 2
TAPEZ UN NOMBRE? 2
TAPEZ UN NOMBRE? 1
NOMBRE DE 1 COMPTES: 3
TAPEZ UN NOMBRE? 410
TAPEZ UN NOMBRE?
```

Examinons ce programme. Les instructions 10 à 20 affichent le message :

```
10  PRINT"JE VAIS COMPTER LE NOMBRE DE 1 TAPES"
20  PRINT"TAPEZ 0 POUR ARRETER"
```

L'instruction 30 initialise la variable compteur SOMME à zéro :

```
30  SOMME=0
```

Le nombre est ensuite entré au clavier :

```
40  INPUT"TAPEZ UN NOMBRE";NOMBRE
```

Si le nombre est 0, le test

```
50  IF NOMBRE=0 THEN 100
```

est positif. L'instruction 100 (END) est exécutée. Supposons que le nombre soit 10, le test :

```
60  IF NOMBRE< >1 THEN GOTO 40
```

est positif. Nous revenons alors à l'instruction 40 et demandons un nouveau nombre. Si le nombre est 1, l'instruction :

```
70  SOMME=SOMME+1
```

est exécutée. Le compteur marque une unité de plus. Rappelons la signification d'une instruction d'affectation. Vous pouvez lire l'instruction 70 comme :

SOMME prend la valeur (ancienne valeur de SOMME) + 1

Ici, SOMME prend la valeur 0 + 1 = 1. L'instruction suivante est :

```
80  PRINT"NOMBRE DE 1 COMPTES:";SOMME
```

Puis le programme boucle sur l'instruction 40 et demande un nouveau nombre :

```
90  GOTO 40
```

Exercice d'arithmétique révisé

Rappelons que nous avons mis au point au début de ce chapitre un programme d'exercice arithmétique. Nous avons regretté qu'il soit trop simple et ne puisse être bouclé sur lui-même. Nous pouvons à présent l'améliorer :

Le programme étant plutôt long, voyons seulement les parties qui nous intéressent. Voici d'abord la partie où l'utilisateur doit choisir un nombre entre 1 et 4 :

```
100  INPUT"VOTRE CHOIX:";CHOIX
110  IF (CHOIX=1) THEN 200
```

```
120  IF (CHOIX=2) THEN 300
130  IF (CHOIX=3) THEN 400
140  IF (CHOIX=4) THEN 500
150  PRINT"IL FAUT CHOISIR UN NOMBRE ENTRE"
160  PRINT"1 ET 4. AU REVOIR !!!":END
```

Nous allons uniquement changer l'instruction 160.

```
160  GOTO 90
```

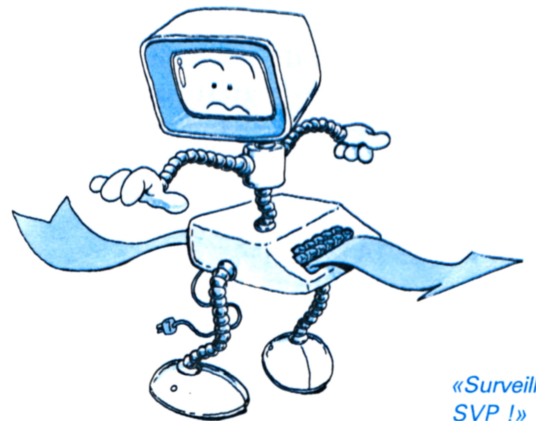
Faites la vérification.

Maintenant nous aimerions que le programme offre non pas une seule question mais dix questions différentes. Cela est possible en ajoutant l'instruction GOTO et un compteur.

Validation de l'entrée (INPUT)

Lorsque l'on entre des données au clavier, on ne peut s'attendre à ce qu'elles soient toujours correctement tapées. Il arrivera à l'utilisateur de faire une erreur de frappe. Il est donc fortement conseillé de *valider* chaque entrée, c'est-à-dire de la vérifier afin, si elle n'est pas correcte, d'être en mesure de la refuser en réclamant une nouvelle entrée. Dans la plupart de nos exemples, nous validerons les entrées.

Prenons deux programmes complets capables de prendre des décisions.



«Surveillez mon entrée SVP !»

Conversion de miles en kilomètres

Au Chapitre 3, nous avons appris à convertir des miles en kilomètres. En voici la version automatisée :

```
10  REM ** CONVERSION DE MILES EN KILOMETRES **
20  REM
30  PRINT"JE CONVERTIS DES MILES EN KILOMETRES"
40  PRINT"TAPEZ 0 POUR ARRETER"
50  INPUT"COMBIEN DE MILES";MILES
60  IF MILES=0 GOTO 100
70  KM=MILES*1.6
80  PRINT MILES;"MILES =";KM;"KILOMETRES"
90  GOTO 50
100 END
```

Nous obtiendrons par exemple :

```
RUN
JE CONVERTIS DES MILES EN KILOMETRES
TAPEZ 0 POUR ARRETER
COMBIEN DE MILES? 7
7 MILES = 11.2 KILOMETRES
COMBIEN DE MILES? 10
10 MILES = 16 KILOMETRES
COMBIEN DE MILES? 0
Ready
```



Anniversaire

Voici un autre exemple : nous allons améliorer notre premier programme de calcul de l'âge d'une personne. Vous allez donner la date de votre naissance (le jour, le mois, l'année) et la date du jour (le jour, le mois, l'année) et le programme calculera votre âge exact.

```
10  REM **CALCUL DE L'AGE**
20  INPUT"DONNEZ VOTRE PRENOM";PRENOM$
30  PRINT"HELLO ";PRENOM$;" JE VAIS"
40  PRINT"CALCULER VOTRE AGE"
45  PRINT"DATE D'AUJOURD'HUI          (JJ/MM/AA)"
50  INPUT"LE JOUR:";JJ
60  IF (JJ<1 OR JJ>31) THEN 50
70  INPUT"LE MOIS (1 A 12):";MM
80  IF (MM<1 OR MM>12) THEN 70
90  INPUT"ANNEE (2 CHIFFRES):";AA
100 IF (AA<0 OR AA>99) THEN 90
110 REM
120 PRINT"DATE DE NAISSANCE"
130 INPUT"LE JOUR:";JNAIS
140 IF (JNAIS<1 OR JNAIS>31) THEN 130
150 INPUT"LE MOIS:";MNAIS
160 IF (MNAIS<1 OR MNAIS>12) THEN 150
170 INPUT"L'ANNEE:";ANAIS
180 IF (ANAIS<0 OR ANAIS>99) THEN 170
190 REM
200 REM ---------CALCUL DE L'AGE---------
210 IF MNAIS<MM THEN 270
220 IF MNAIS>MM THEN 320
230 REM ---ANNIVERSAIRE CE MOIS-CI---
240 IF JNAIS<JJ THEN 270
250 IF JNAIS>JJ THEN 320
260 PRINT"C'EST AUJOURD'HUI VOTRE ANNIVERSAIRE!!"
270 AGE=AA-ANAIS
280 PRINT"VOUS AVEZ ";AGE;" ANS"
290 END
300 REM --ANNIVERSAIRE PAS ENCORE ATTEINT--
```

```
320  AGE=AA-ANAIS-1
330  GOTO 280
340  END
```

En dépit de sa longueur, ce programme est en fait très simple. Notons la façon de valider chaque entrée (INPUT). Toutefois, pour que notre programme reste court, notre validation est plutôt sommaire. Nous ne vérifions pas que chaque nombre est un entier, ni le nombre de jours dans chaque mois. Ce sera un bon exercice pour le lecteur consciencieux et patient... Voici une façon de vérifier que M est égal à un entier compris entre 1 et 12 :

```
IF NOT (MM = 1 OR MM = 2 OR... MM = 12) THEN 70
```

Résumé

Les instructions IF et GOTO nous ont permis d'apprendre à écrire des programmes qui exécutent des tests sur des valeurs et qui prennent des décisions. Nous avons aussi appris à faire des boucles de programme de sorte qu'une partie de programme puisse être répétée indéfiniment. En outre, nous avons appris à vérifier et à valider systématiquement les entrées au clavier. Nous avons vu toutes les techniques de base nécessaires pour écrire des programmes courants, et examiné plusieurs exemples significatifs. Maintenant nous allons faire en sorte que nos programmes soient plus aisés à écrire.

En raison de la fréquence et de l'importance des boucles et de l'automatisation dans les programmes, le BASIC nous offre des méthodes de traitement supplémentaires, que nous étudierons dans le prochain chapitre.

Exercices

6-1 : A quoi sert l'instruction IF ?

6-2 : Expliquez les instructions suivantes :

```
10  INPUT REP$
20  IF (REP$ = "OUI") THEN PRINT "MERCI"
30  IF (REP$ = "NON") THEN PRINT "INCORRECT"
40  PRINT"OUI OU NON" : GOTO 10
```

Si le résultat est illogique, suggérez un meilleur programme.

6-3 : Les expressions logiques suivantes sont-elles correctes ?

a. A = 4

b. B = 2 OR C = 3

c. A > 5

d. 5 > A

e. 1 > 2

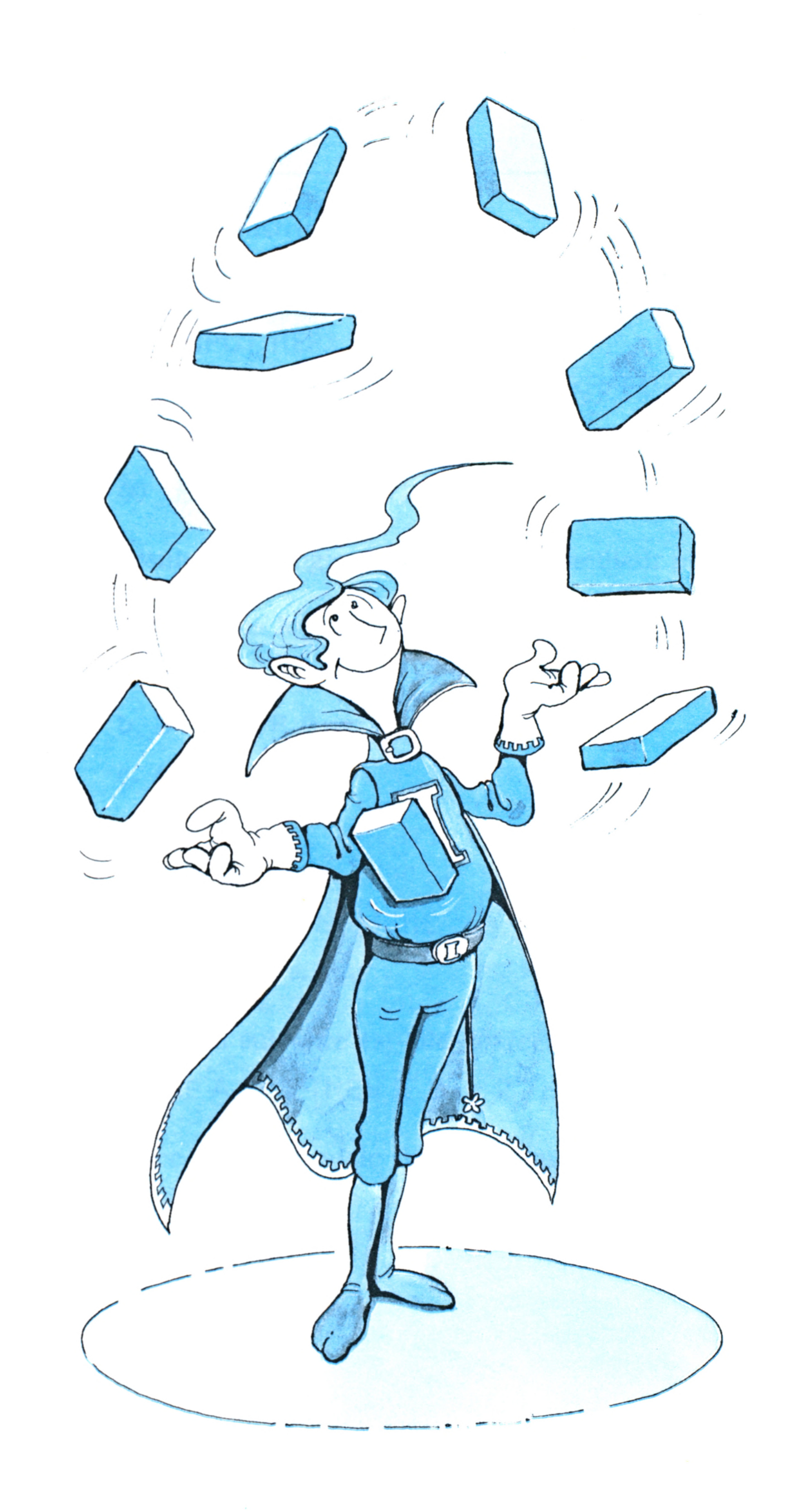
f. VALEUR > NOMBRE

g. LETTRE$ = "A"

6-4 : L'instruction suivante est-elle correcte ?

```
10  IF A = 5 THEN IF B = 2 THEN 18
```

6-5 : Qu'est-ce qu'une boucle de programme ?



7

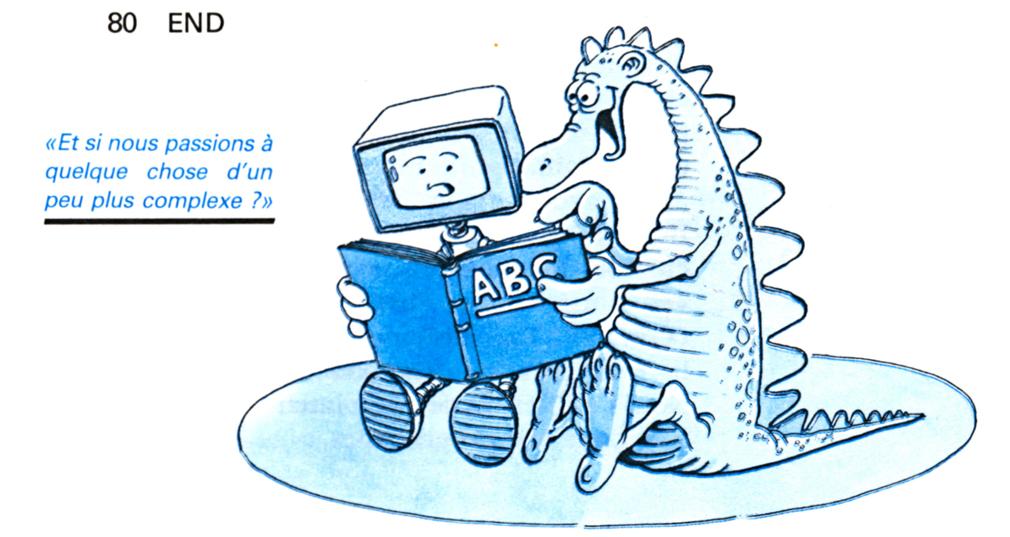
AUTOMATISATION DES REPETITIONS

Nous pouvons réutiliser indéfiniment une partie de programme en nous servant des instructions IF et GOTO. Cette partie du programme s'appellera une boucle. La plupart des programmes comportent des boucles. Dans ce chapitre, nous allons apprendre de nouvelles techniques pour créer des boucles. Nous allons aussi mettre au point des programmes élaborés qui automatisent les tâches. Après avoir revu la technique IF/GOTO pour générer une boucle nous introduirons une nouvelle instruction, l'instruction FOR...NEXT, définie pour faciliter la création des boucles. Nous l'utiliserons de façon intensive dans nos programmes.

La technique IF/GOTO

Commençons par examiner un programme qui automatise une boucle en utilisant la technique IF/GOTO. En examinant le programme ci-dessous, nous dégagerons certaines caractéristiques communes à toutes les boucles. Par exemple, nous verrons les emplois de la variable compteur : incrémentation, initialisation et test avant la sortie. Voici le programme, il calcule la somme des dix premiers nombres entiers.

```
1  REM ***SOMME DES DIX PREMIERS NOMBRES ENTIERS***
10 SOMME=0
20 I=1
30 SOMME=SOMME+I
40 I=I+1
50 IF I=11 THEN 70
60 GOTO 30
70 PRINT"LA SOMME DES DIX PREMIERS NOMBRES"
75 PRINT"ENTIERS EST:";SOMME
80 END
```



«Et si nous passions à quelque chose d'un peu plus complexe ?»

Ce programme fait appel à deux variables : SOMME et I. La variable SOMME accumule la somme des dix premiers nombres entiers à mesure que vous les additionnez - c'est l'équivalent du sous-total d'une machine à calculer. I est l'entier que l'on ajoute à la SOMME.

Rappelons qu'avant d'utiliser pour la première fois une variable on doit lui attribuer une valeur. Donc avant d'utiliser les variables SOMME et I dans une formule, il faut les *initialiser,* c'est-à-dire leur donner une valeur initiale (ici 0 et 1 respectivement). On le fera au moyen des instructions 10 et 20. Ces instructions s'appellent *instructions d'initialisation.*



«Vous me reconnaissez ?»

L'instruction suivante est :

```
30   SOMME=SOMME+I
```

Cette instruction permet d'additionner la valeur de l'entier courant et la variable SOMME courante. Quand cette instruction est exécutée pour la première fois, la valeur SOMME est 0 et celle de I est 1. Il en résulte que cette instruction affecte la valeur 0 + 1 = 1 à la valeur SOMME. Après l'exécution de cette instruction, SOMME contient la valeur 1. L'instruction suivante est :

```
40   I=I+1
```

La valeur courante de I est 1. Le but de cette instruction est d'attribuer à I la nouvelle valeur 2. C'est la technique du compteur : I est incrémenté de 1 pour que l'entier suivant soit généré. En même temps, la valeur de I indique combien de fois les nombres entiers ont été additionnés jusqu'alors. Autrement dit, I est utilisé comme entier courant et comme compteur. Ainsi, il ne nous reste plus qu'à revenir à



«Je vous ai eu encore une fois !»

l'instruction 30 et continuer à additionner les entiers :

```
50  GOTO 30
```

Erreur ! Ce programme (en théorie) ne va jamais s'arrêter (en réalité, il s'arrêtera lorsque la valeur de SOMME sera supérieure au plus grand nombre autorisé par l'interpréteur de l'Amstrad). Mais ce que nous voulons c'est que le programme s'arrête après avoir exécuté la boucle dix fois. Nous devons donc introduire une instruction de *test*. La voici :

```
50  IF I=11 THEN 70
```

Quand I atteint la valeur 11, l'instruction 70 est exécutée et le programme s'arrête. Il s'agit là d'une *sortie de boucle*. Vérifions maintenant que la valeur 11 (plutôt que 10) est réellement correcte dans l'instruction 50. Si nous écrivons l'instruction :

```
50  IF I=10 THEN 70
```

nous nous apercevons que l'exécution est incomplète. En effet, quand I atteint la valeur 10, la variable SOMME contient la somme des 9 premiers chiffres seulement. La boucle doit être exécutée une fois de plus. Rappelons que chaque boucle contient un compteur. Vérifiez avec soin la valeur du compteur qui déclenche la sortie hors de la boucle. Dans notre exemple, tant que I n'est pas égal à 11, la boucle sera à nouveau exécutée.

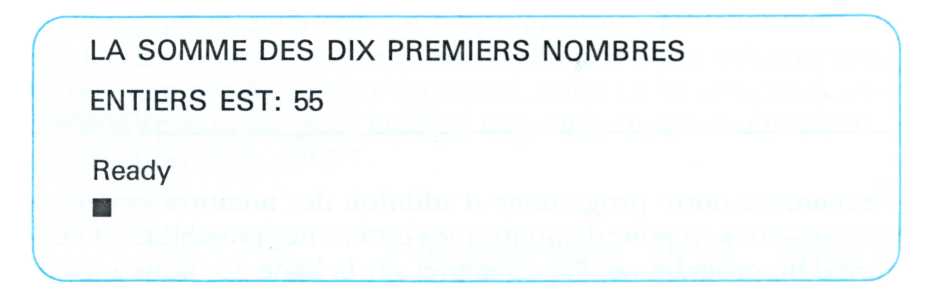
```
60  GOTO 30
```

Quand I atteint la valeur 11, les dix premiers nombres entiers sont additionnés. Ceci parce que dans notre programme, l'addition (SOMME = SOMME + 1) a lieu avant l'incrémentation (I = I + 1). Les trois instructions finales du programme sont les instructions de sortie hors de la boucle :

```
70  PRINT"LA SOMME DES DIX PREMIERS NOMBRES"
75  PRINT"ENTIERS EST:";SOMME
```

```
80  END
```

Après avoir lancé ce programme, nous obtiendrons :



```
LA SOMME DES DIX PREMIERS NOMBRES
ENTIERS EST: 55

Ready
■
```

La Figure 7.1 représente la suite des commandes dans le programme. Les nombres entre parenthèses correspondent aux numéros d'instructions.

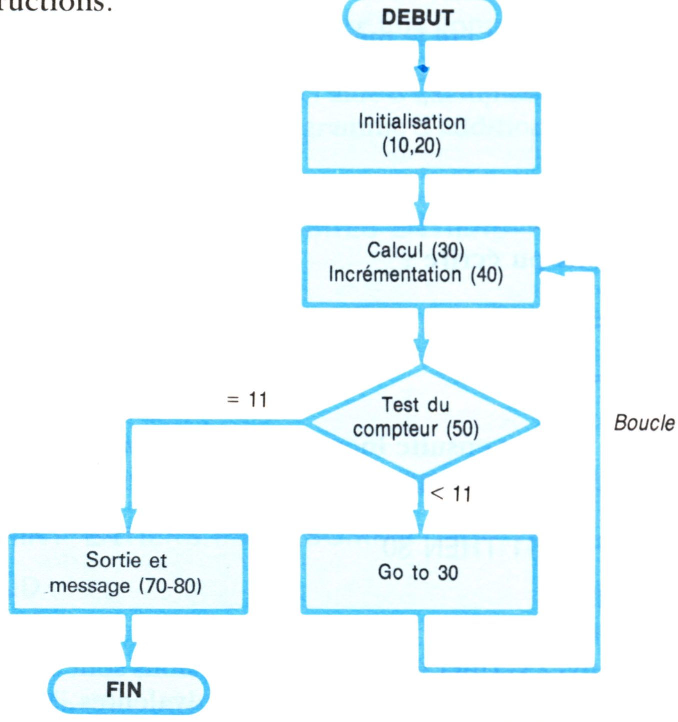


Figure 7.1 : Organigramme du programme SOMME

Ce diagramme s'appelle un *organigramme* ; nous allons l'étudier en détail au Chapitre 8. Pour le moment, notons tout simplement l'organisation générale du programme : initialisation, calcul plus incrémentation, test et sortie. Toutes les parties du programme comportant des boucles exécutent ces fonctions.

Variantes

Revenons à notre programme d'addition des nombres entiers et servons-nous en pour démontrer les différentes possibilités d'écriture d'un programme. Par exemple, sur la ligne 50, nous aurions pu écrire :

```
50  IF I > 10 THEN 70
```

et le résultat aurait été le même (quand I atteint la valeur 11, elle est supérieure à 10). Au lieu de :

```
40  I = I + 1
50  IF I = 11 THEN 70
```

Nous aurions pu écrire :

```
40  IF I = 10 THEN 70
50  I = I + 1
```

I est alors testé et ensuite incrémenté. Notons que cette fois I est testé pour la valeur 10 (au lieu de 11). Nous aurions pu écrire aussi :

```
50  IF I < 11 THEN 30
60  REM
```

Vous pouvez vérifier que toutes ces versions sont correctes. Toutes ces variantes sont acceptables et équivalentes. Même un petit programme comme le programme SOMME peut s'écrire de diffé-

rentes façons, toutes équivalentes. En fait, il n'existe pas une seule manière d'écrire un programme. Un concept peut s'exprimer sous différentes formes comme dans le langage parlé.

Ce court programme est une illustration de l'emploi de la boucle et de la variable compteur. Nous avons aussi examiné les phases types qui caractérisent un tel programme : initialisation, calcul, incrémentation, test et sortie. Les boucles sont utilisées dans les programmes de manière intensive ; aussi a-t-on conçu une instruction appropriée pour faciliter leur mise en œuvre en BASIC : l'instruction FOR...NEXT.

L'instruction FOR...NEXT (POUR...SUIVANT)

L'instruction FOR... NEXT automatise une bonne partie de la programmation requise pour une boucle. Nous allons expliquer son utilisation et son fonctionnement à l'aide d'exemples concrets. Voici une manière de réécrire notre programme d'addition en appliquant cette instruction :

```
1  REM **ADDITION NOMBRES ENTIERS—V2**
10 SOMME=0
20 FOR I=1 TO 10
30 SOMME=SOMME+I
40 NEXT I
50 PRINT"LA SOMME DES DIX PREMIERS NOMBRES"
55 PRINT"ENTIERS EST:";SOMME
60 END
```

Vous remarquerez que ce programme comporte deux instructions de moins que le précédent. Il est plus court et plus lisible.

Examinons-le en détail. La première instruction exécutable initialise la variable SOMME à zéro :

```
10  SOMME=0
```

L'instruction suivante est l'instruction FOR :

```
20  FOR I=1 TO 10
```

Cette instruction joue plusieurs rôles :

— elle marque le début de la boucle automatique (c'est là que commence la boucle).

— elle spécifie que I (la variable compteur) commence par la valeur initiale 1 la première fois que cette instruction est exécutée. Ainsi n'est-il pas nécessaire de créer une instruction d'initialisation pour I.

— I est incrémenté de 1 (jusqu'à la valeur maximum 10) chaque fois que l'instruction est réactivée par une instruction NEXT correspondante. Le test est exécuté automatiquement, et quand I dépasse la valeur 10, la boucle se termine et l'instruction qui suit NEXT est exécutée (il s'agit là de la sortie de la boucle).

Le corps de la boucle contient tout simplement le cumul de la somme :

```
30  SOMME=SOMME+I
```

L'instruction suivante :

```
40  NEXT I
```

marque la fin de la boucle et entraîne la réactivation de l'instruction FOR. Cette instruction équivaut à deux instructions dans la version initiale.

```
40  I=I+1
60  GOTO 30
```

Chaque fois que l'instruction NEXT I est exécutée, le programme saute au début de la boucle, c'est-à-dire à l'instruction FOR. Quand FOR est activé :

— I est incrémenté de 1

— la nouvelle valeur de I est automatiquement comparée à 10.

Tant que I ne dépasse pas 10, l'exécution se poursuit. La boucle s'arrête quand I est égal à 10 et NEXT est atteint. A ce moment-là, on sort de la boucle, et l'instruction 50 (après NEXT) est exécutée. Cette séquence est représentée à la Figure 7.2 (organigramme). La Figure 7.2 montre que l'instruction FOR automatise trois tâches :

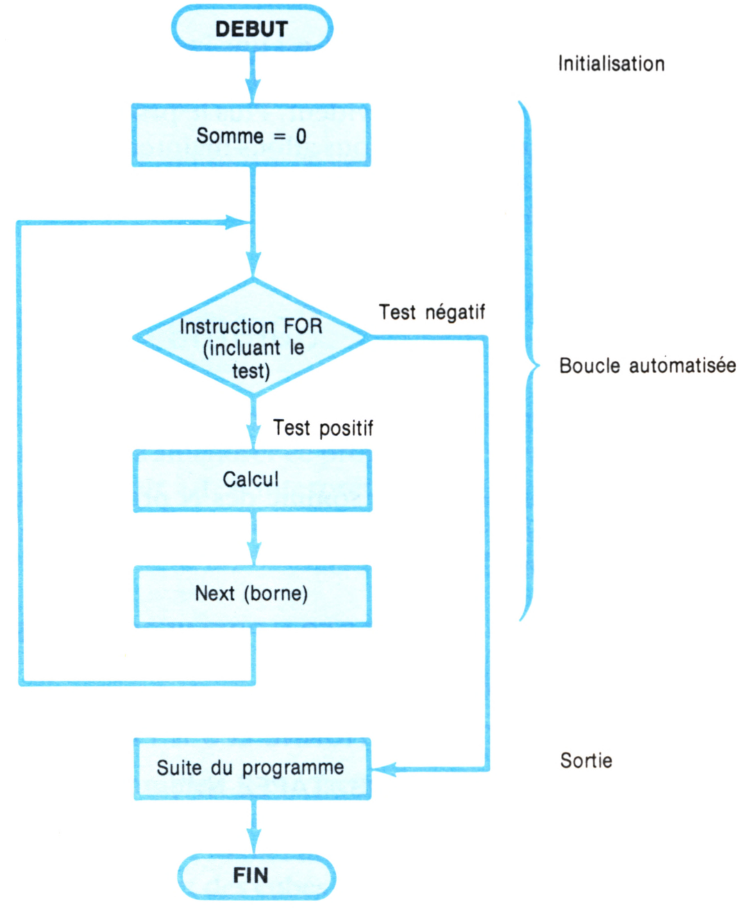


Figure 7.2 : Boucle automatique utilisant FOR...NEXT

— initialisation du compteur (I est initialisé à 1)
— incrémentation du compteur (I est incrémenté de 1 à chaque fois)
— test du compteur par rapport à une valeur maximum (ici I est comparé à 10).

L'instruction NEXT marque tout simplement la fin de la boucle et donne lieu à l'exécution d'un «GOTO instruction FOR». Quand vous aurez manipulé l'instruction FOR plusieurs fois, vous vous rendrez compte qu'elle simplifie la définition de la boucle et clarifie le programme.

L'instruction FOR...NEXT est une instruction très pratique. Vous n'êtes pas tenu de vous en servir, mais vous allez probablement la trouver d'un intérêt évident. Plus le programme est clair, moins il y a risque d'erreurs. Nous allons maintenant illustrer les possibilités des instructions FOR...NEXT et des boucles automatiques à l'aide de quelques exemples.

Somme des N premiers nombres entiers

Nous allons calculer la somme des N premiers nombres entiers. Cette fois-ci, l'utilisateur définit la valeur de N au clavier. Voici le programme :

```
10 REM **SOMME DES N PREMIERS NOMBRES ENTIERS**
20 SOMME=0
25 PRINT"J'ADDITIONNE LES N PREMIERS NOMBRES"
30 PRINT"ENTIERS. TAPEZ N:";
35 INPUT N
40 FOR I=1 TO N
50 SOMME=SOMME+I
```

```
60  NEXT I
70  PRINT"LA SOMME DES ";N;"PREMIERS
    NOMBRES EST:";SOMME
80  END
```

Vous obtiendrez (par exemple) :

```
RUN
J'ADDITIONNE LES N PREMIERS NOMBRES
ENTIERS. TAPEZ N:? 5
LA SOMME DES 5 PREMIERS NOMBRES EST: 15
Ready
■
```

Cette fois nous bouclons de 1 à N, N étant entré au clavier (instruction 35). Vous pouvez améliorer ce programme en validant l'entrée : N doit être supérieur à 1. L'interpréteur BASIC va automatiquement vérifier si N est un nombre entier ou non lors de l'exécution de l'instruction FOR. Essayez de le contrecarrer.

Tables de valeurs

Nous allons montrer combien il est facile d'automatiser les calculs et d'imprimer des tables de valeurs en nous servant de la puissante instruction FOR...NEXT. Voici une table des carrés (un nombre multiplié par lui-même) et des cubes (un nombre multiplié deux fois par lui-même) :

```
10  REM *TABLE DES CARRES ET DES CUBES*
20  REM *POUR LES 10 PREMIERS NOMBRES ENTIERS*
30  FOR I=1 TO 10
40  PRINT I,I*I,I*I*I
50  NEXT I
60  END
```

Voici le résultat :

```
RUN
 1        1        1
 2        4        8
 3        9        27
 4        16       64
 5        25       125
 6        36       216
 7        49       343
 8        64       512
 9        81       729
 10       100      1000
```

Voyons de plus près l'instruction 40 :

```
40  PRINT I,I*I,I*I*I
```

I*I signifie I à la puissance 2. Par exemple, si I = 2 alors I*I = 2 × 2 = 4. De même, I*I*I signifie I à la puissance 3. Si I = 4, I*I*I = 4 × 4 × 4 = 64.

Vous remarquerez que nous utilisons une virgule dans l'instruction PRINT pour que les colonnes soient alignées.

Comme exercice, nous vous proposons de réécrire ce programme pour afficher la somme des carrés et des cubes des N premiers nombres entiers, N étant entré au clavier. Nous avons appris à le faire dans le paragraphe précédent.

Lignes d'étoiles

Voici un programme simple qui imprime un nombe N de lignes d'étoiles, N étant un nombre défini au clavier.

```
10 REM **LIGNES D'ETOILES**
20 PRINT"JE VAIS AFFICHER DES LIGNES D'ETOILES"
30 INPUT"COMBIEN EN VOULEZ-VOUS:";N
40 REM N=NOMBRE D'ETOILES
50 FOR I=1 TO N
60 PRINT"*************************"
70 NEXT I
80 END
```

Nous obtiendrons (par exemple) :

```
RUN
JE VAIS AFFICHER DES LIGNES D'ETOILES
COMBIEN EN VOULEZ-VOUS:? 6
*************************
*************************
*************************
*************************
*************************
*************************
```

Répétons que chaque fois que l'utilisateur fait une entrée (INPUT), il doit, pour éviter toute mauvaise surprise, la valider. Ici, le nombre demandé devra être positif. Supposons que vous ne vouliez pas obtenir plus de 20 lignes d'étoiles ; il faudra en informer l'utilisateur au moyen d'une instruction PRINT et utiliser une instruction de validation telle que :

```
IF (N < 1) OR (N > 20) GOTO 20
```

Les boucles élaborées

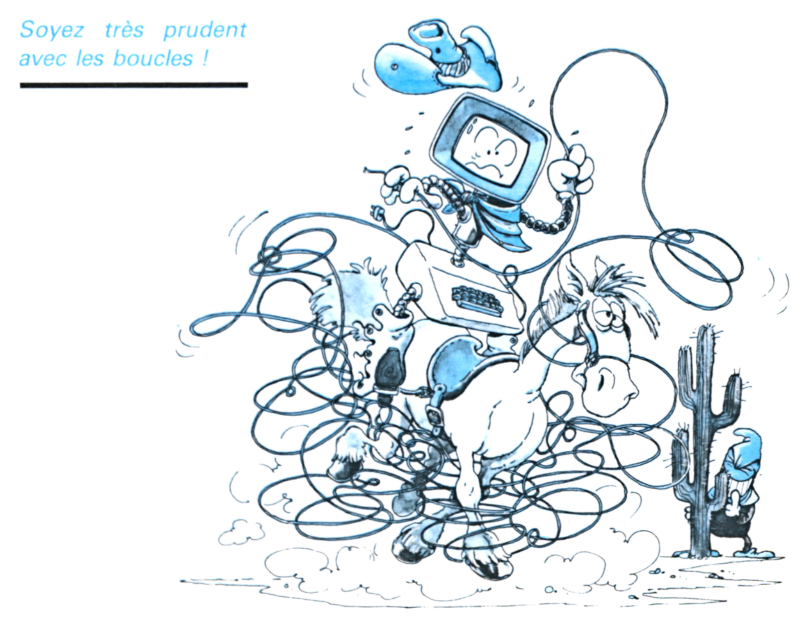
L'instruction FOR...NEXT offre deux possibilités particulières que nous n'avons pas encore décrites :

— Vous pouvez incrémenter le compteur à l'aide de n'importe quelle valeur entière telle que 2, 3, 4 ou même − 1. Il s'agit là du *pas variable.*

— Vous pouvez créer une boucle à l'intérieur d'une autre boucle. De telles boucles s'appellent des *boucles imbriquées.*

Examinons successivement ces deux possibilités :

Soyez très prudent avec les boucles !



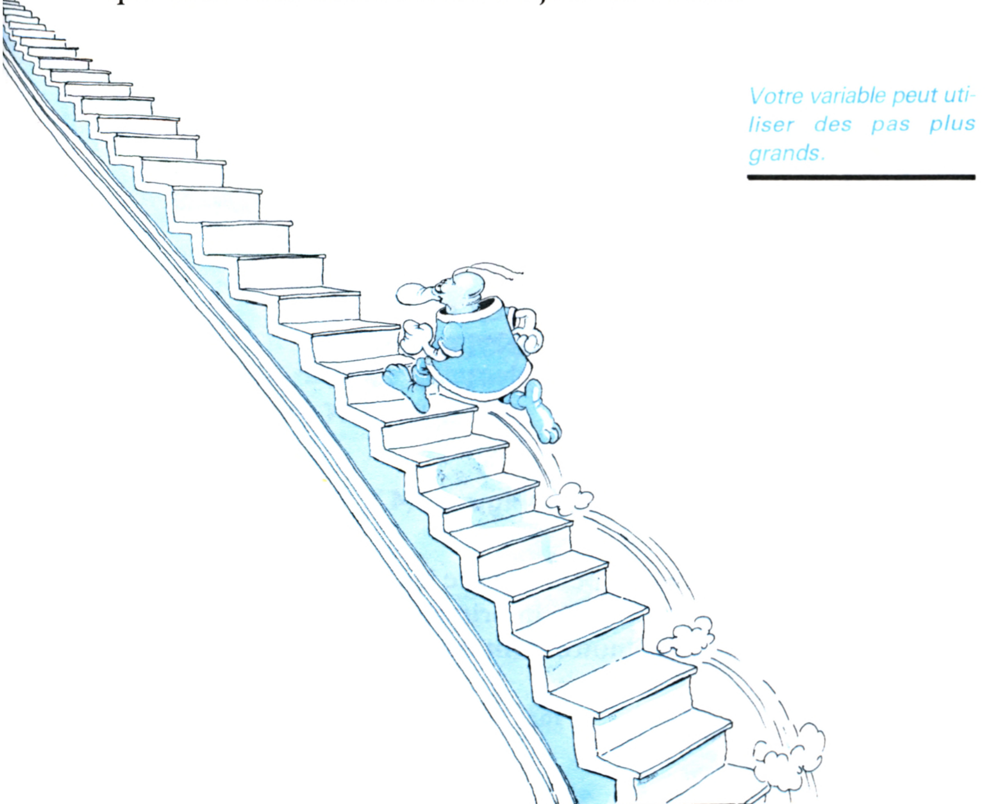
En voici un exemple :

```
FOR I = 1 TO 5 STEP 2
```

Chaque fois que la boucle est exécutée, I est incrémenté de 2. Vous pouvez même écrire une instruction telle que :

```
FOR I = 10 TO – 5 STEP – 1
```

qui utilise pour incrément un *pas négatif.* La limite supérieure du compteur (– 5 ici) étant inférieure à la valeur de départ (10) il s'agit là, pour l'interpréteur, d'un «pas négatif». La valeur de I sera donc, à chaque fois, diminuée de 1. La première valeur de I sera 10, la suivante 9, puis 8, etc. et enfin – 5 la dernière. Autrement dit, I prendra successivement les valeurs 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, – 1, – 2, – 3, – 4, et – 5. Le pas négatif est une fonction pratique dont vous aurez besoin un jour ou l'autre.

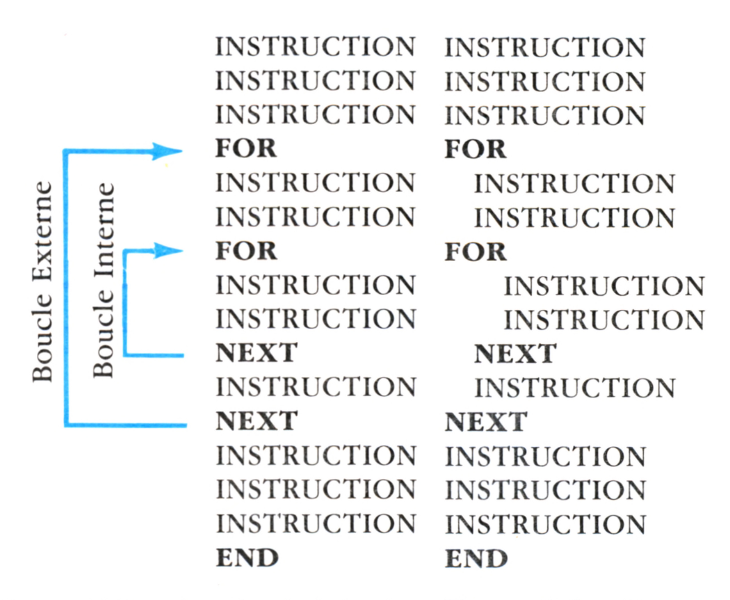


Votre variable peut utiliser des pas plus grands.

La technique des boucles imbriquées est une méthode d'un très grand intérêt utilisée pour automatiser des traitements complexes. Une boucle imbriquée est créée chaque fois que vous utilisez un groupe d'instructions FOR...NEXT à l'intérieur d'une boucle.

Vous pouvez généralement insérer autant d'instructions que vous voulez entre les instructions FOR et NEXT. Vous pouvez même inclure une autre boucle à l'intérieur de ces instructions. C'est alors une boucle imbriquée. Voir Figure 7.3.

Notons qu'il est difficile de lire un programme comportant des boucles imbriquées. Pour remédier à cet inconvénient, nous vous conseillons *le décalage*, autre technique de clarification d'un programme. La Figure 7.4 représente une version décalée du programme de la Figure 7.3.

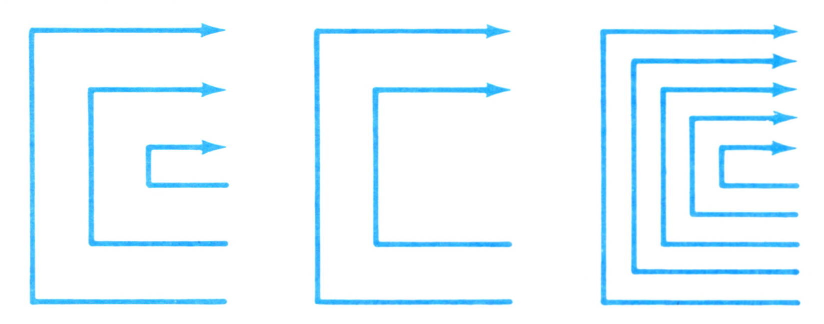



```
INSTRUCTION
INSTRUCTION
INSTRUCTION
FOR
  INSTRUCTION
  INSTRUCTION
FOR
    INSTRUCTION
    INSTRUCTION
  NEXT
  INSTRUCTION
NEXT
INSTRUCTION
INSTRUCTION
INSTRUCTION
END
```

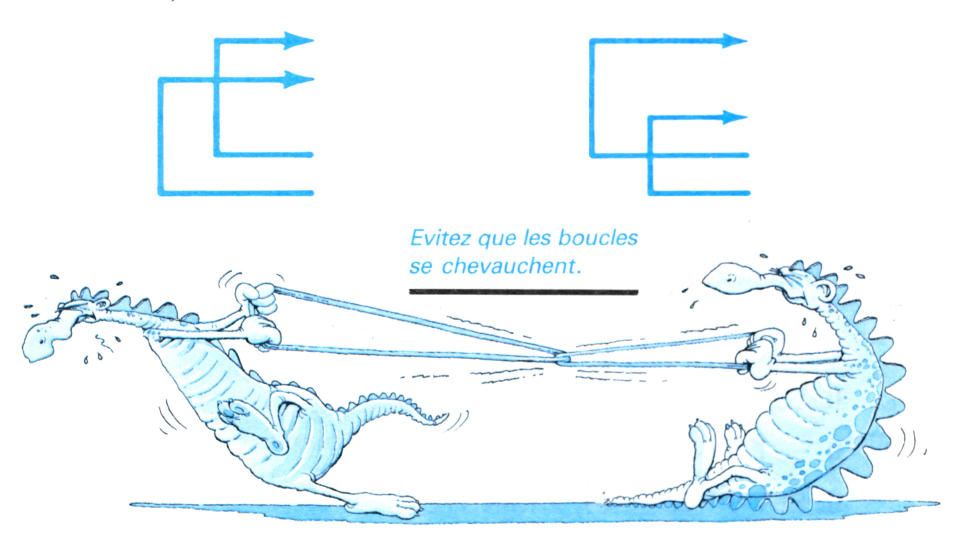
Figure 7.3. : Boucles imbriquées **Figure 7.4.** : Programme indenté

L'imbrication des boucles peut se faire à n'importe quel niveau ; le nombre maximum de boucles imbriquées dépend de l'interpréteur ou de la capacité mémoire dont on dispose. Toutefois, il ne

peut y avoir chevauchement de deux boucles. Les boucles ci-dessous sont correctes :



Par contre, celles-ci sont incorrectes :



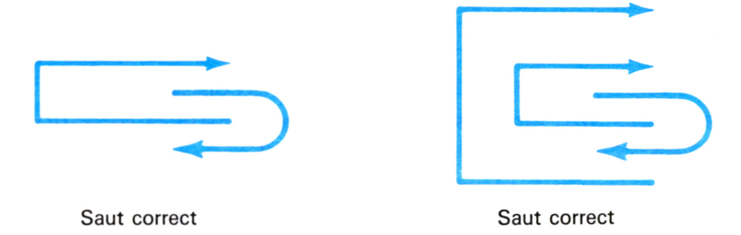
De plus, on ne peut pas sauter (c'est-à-dire utiliser un GOTO) d'un point *à l'intérieur* de la boucle extérieure vers un point à l'intérieur de la boucle intérieure :



Imbrication correcte

Saut interdit (avec IF ou GOTO)

On peut, par contre, sauter hors de la boucle intérieure :



Saut correct Saut correct

Voici un exemple de boucle imbriquée. Ce programme affiche le temps accéléré en heures et en minutes :

```
10 REM **HORLOGE SIMULEE**
20 FOR HEURE=0 TO 23
30 FOR MINUTE=0 TO 59
40 PRINT"IL EST";HEURE;"HEURE(S),"
   ;MINUTE;"MINUTE(S)"
50 NEXT MINUTE
60 NEXT HEURE
70 PRINT"FIN DE LA JOURNEE"
80 END
```

Vous obtenez :

```
RUN
IL EST 0 HEURE(S), 0 MINUTE(S)
IL EST 0 HEURE(S), 1 MINUTE(S)
IL EST 0 HEURE(S), 2 MINUTE(S)
IL EST 0 HEURE(S), 3 MINUTE(S)
IL EST 0 HEURE(S), 4 MINUTE(S)
            .
            .
            .
IL EST 0 HEURE(S), 59 MINUTE(S)
IL EST 1 HEURE(S), 0 MINUTE(S)
```

Caractéristiques supplémentaires

Pour terminer, nous dirons que les valeurs décimales et les expressions sont en général autorisées dans l'instruction FOR. Par exemple, voici des instructions correctes :

```
FOR MESURE = 0.1 TO 13.5 STEP 0.2
FOR VALEUR = N TO (N * 2) STEP 1
```

Nous ne vous recommandons pas d'utiliser de telles instructions qui sont assez compliquées. Nous ne les emploierons pas ici.

Résumé

Les boucles sont très utilisées pour automatiser la répétition d'une partie d'un programme. En BASIC on se sert de l'instruction FOR...NEXT qui peut souvent remplacer d'autres instructions BASIC. Dans ce chapitre, nous avons étudié les différents usages de cette instruction y compris les boucles imbriquées, et nous avons mis au point plusieurs programmes plus élaborés. Maintenant que vous connaissez les principales techniques de programmation, vous allez pouvoir commencer à écrire vos propres programmes. Dans le prochain chapitre, nous vous aiderons à faire vos premiers pas.



Exercices

7-1 : Affichez les 15 premiers nombres entiers sur une ligne (4 instructions).

7-2 : Ecrivez un programme qui lit le temps en heures et en minutes au clavier et l'affiche comme suit :

Entrée :	3 *(heures),*	11 *(minutes)*
Affichage :	HHH	*(3 lettres)*
	MMMMMMMMMMM	*(11 lettres)*

7-3 : Pouvez-vous sauter au milieu d'une boucle ?

7-4 : Calculez la somme des N premiers nombres impairs - N étant entré au clavier - et affichez-la pour chaque entier.

7-5 : Affichez une table de T.V.A. pour des prix variant de 10 F à 1000 F avec un incrément de 10 F. Entrez le taux de la T.V.A. au clavier.



8

CREER UN PROGRAMME

La programmation implique la conception d'un programme qui automatise une tâche. Jusqu'à maintenant, nous avons écrit plusieurs petits programmes sans étapes intermédiaires, c'est-à-dire en écrivant directement une suite d'instructions BASIC. Cette technique convient à des programmes très simples, mais elle est insuffisante dès lors qu'il s'agit de programmes plus complexes.

Dans ce chapitre, nous allons étudier la meilleure façon d'écrire un programme. Cette opération comporte cinq phases :

1. Définir la suite des étapes nécessaires pour résoudre le problème. C'est-à-dire définir l'*algorithme.*
2. Tracer un diagramme représentant la suite des événements et les étapes logiques. C'est-à-dire tracer l'*organigramme.*
3. Ecrire le programme en BASIC. C'est le *codage.*
4. Vérifier et tester le programme. C'est la *mise au point.*
5. Clarifier et documenter le programme : c'est la *documentation.*

Jusqu'a présent, nous n'avons exécuté que les phases 2 et 5. Mais si elles suffisent pour les programmes courts, nous devons, avant de poursuivre avec des programmes plus longs, étudier toutes les phases nécessaires à leur conception.

Définition de l'algorithme

Nous voulons réaliser un programme qui résout un problème donné ou automatise une tâche. Jusqu'à présent, nous avons réalisé des programmes pour résoudre des problèmes simples. La succession des étapes était évidente et de ce fait la phase de définition de l'algorithme inexistante. Cependant, de façon générale, un problème étant posé, nous chercherons la suite des opérations à effectuer pour le résoudre. On voit en conséquence que l'*algorithme* se définit comme la séquence des opérations à effectuer (chacune d'elles étant rigoureusement précisée) pour trouver la solution d'un problème. Techniquement, un algorithme ne peut continuer indéfiniment. Un algorithme qui n'a pas de fin ne peut être... qu'une *erreur !*

Voici un problème simple : convertissons un poids exprimé en onces en son équivalent en grammes. Rappelons qu'une once est égale à 28.35 grammes. La solution est évidente : nous devons multiplier le poids en onces par 28.35. C'est là un algorithme simple à une seule étape.

Un peu plus complexe : lisons un nombre au clavier, et déterminons s'il se trouve dans un certain intervalle. Nous considérerons que le nombre est correct s'il est situé entre 0 et 100. La série des étapes nécessaires pour résoudre ce problème est :

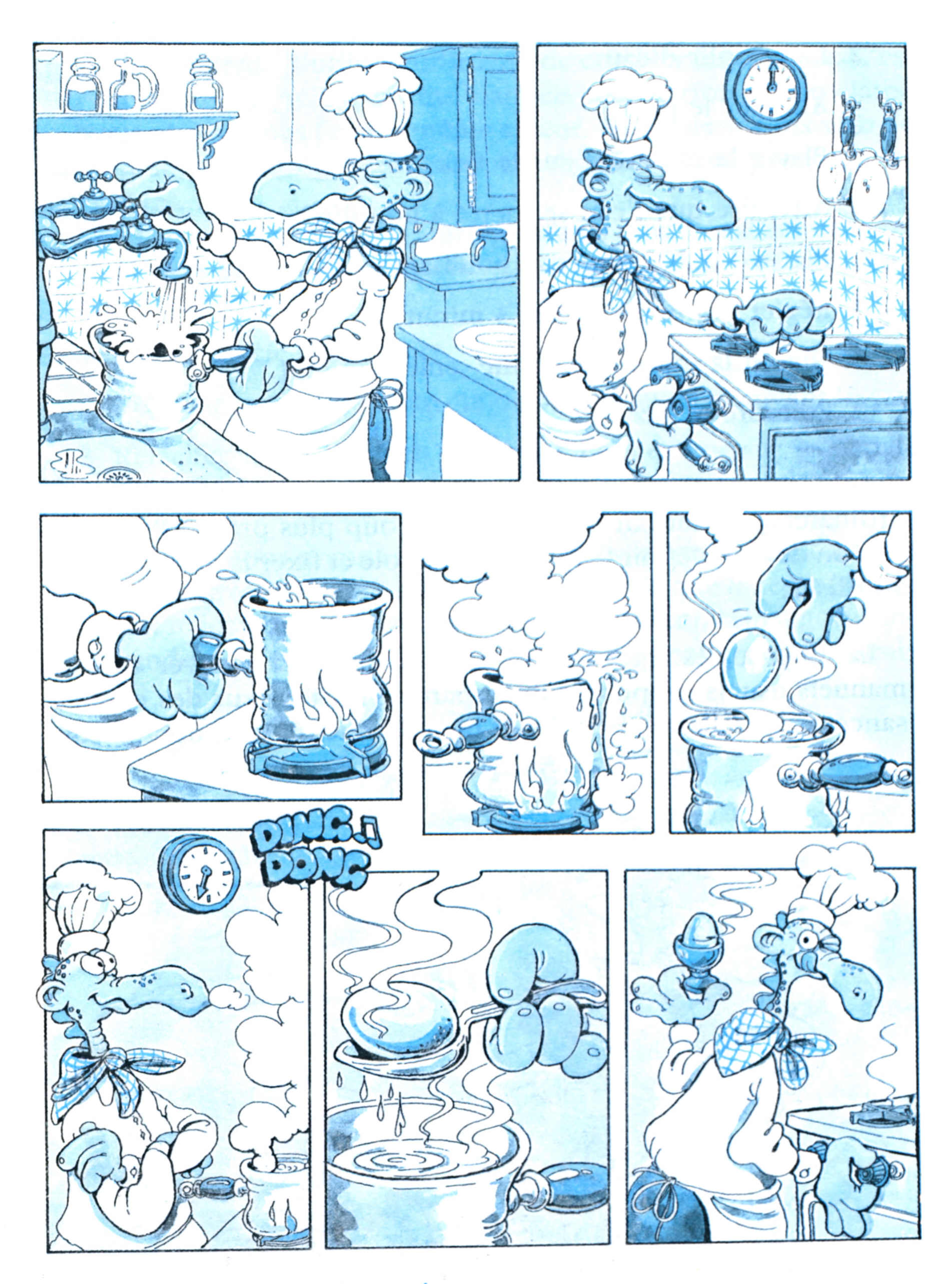
1. Lire le nombre.
2. Vérifier s'il est supérieur à 0. Si oui, continuer ; sinon éliminer le nombre.
3. Vérifier s'il est inférieur à 100. Si oui, accepter le nombre ; sinon le rejeter.

C'est un algorithme à trois étapes.

Dans la pratique, la plupart des problèmes sont plus compliqués. Vous trouverez des algorithmes plus complexes dans votre livre de cuisine ou dans le manuel d'utilisation de votre automobile.

Voici par exemple un algorithme pour la préparation d'un œuf à la coque. Il comprend neuf étapes :

1. Prendre une casserole.



Un algorithme en images : la préparation d'un œuf à la coque.

2. La remplir d'eau.

3. Allumer le gaz.

4. Placer la casserole sur le feu.

5. Attendre que l'eau se mette à bouillir.

6. Placer l'œuf dans l'eau bouillante.

7. Régler la minuterie sur 3 minutes.

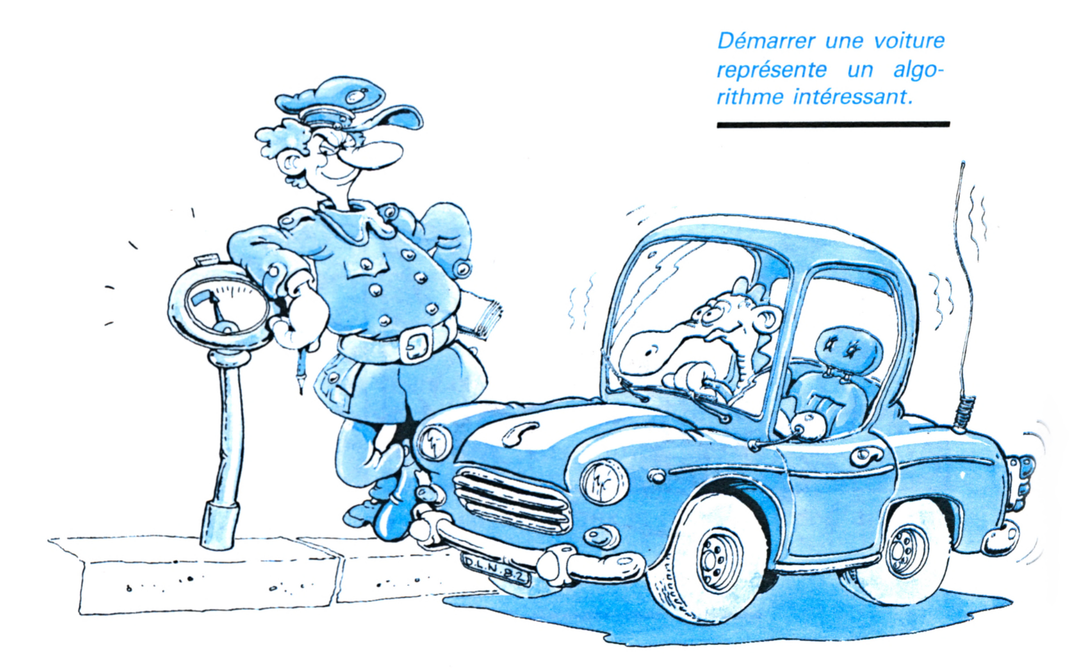
8. Quand la minuterie sonne, enlever l'œuf.

9. Eteindre le gaz.

Cet algorithme paraît clair, mais s'il devait être exécuté par un robot ordinateur, il faudrait qu'il soit beaucoup plus précis. Par exemple, on devrait définir le type de casserole et fixer la quantité d'eau à utiliser.

La plupart des algorithmes que vous pouvez trouver dans les manuels usuels supposent de la part de l'utilisateur des connaissances culturelles et techniques bien définies et sont donc incom-

Démarrer une voiture représente un algorithme intéressant.



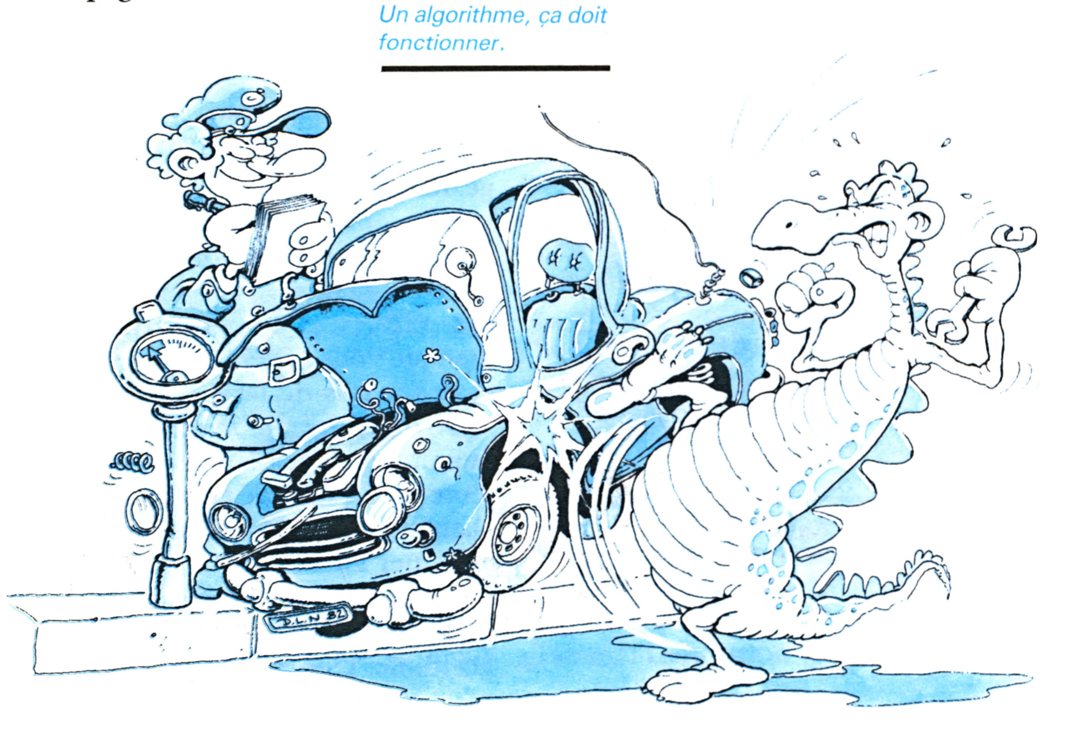
plets ; ils laissent à l'utilisateur le soin de combler les lacunes. C'est la raison pour laquelle certains manuels sont parfois si peu clairs.

Nous n'allons pas faire la même erreur. Nous prendrons soin de bien définir nos algorithmes pour aboutir à des programmes utilisables.

Voici un dernier exemple : un algorithme pour démarrer une voiture. Si nous supposons que cette voiture fonctionne parfaitement, l'algorithme est très simple :

1. Insérer la clé de contact.
2. Tourner la clé vers la droite ou vers la gauche suivant le cas.
3. Relâcher la clé tout en appuyant doucement sur la pédale d'accélération.

Toutefois, nous savons qu'une voiture ne démarre pas toujours à la première tentative, des facteurs divers tels que la température ou l'état mécanique du moteur peuvent intervenir. Un algorithme qui prendrait en compte la totalité des facteurs couvrirait plusieurs pages.



Un algorithme, ça doit fonctionner.

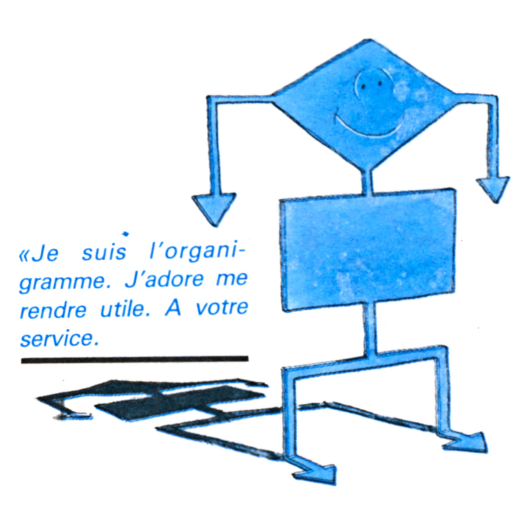
Dans la vie courante, nous pouvons souvent simplifier les étapes d'un algorithme. Mais ce n'est guère possible dans un programme d'ordinateur : un algorithme doit être correct et complet.

Lorsque vous écrivez un programme, vous devez définir l'algorithme avec précision et en prévoyant toutes les circonstances et toutes les éventualités, faute de quoi votre programme ne pourrait être exécuté. Le travail de programmation exige une attitude particulière de constante vigilance et d'esprit critique en éveil. Vous devez à chaque instant considérer que ce que vous avez écrit jusque-là peut comporter une erreur ou une lacune, et par conséquent être toujours prêt à vérifier et à corriger.

En résumé, pour automatiser la solution d'un problème commencez par préparer un algorithme. Sous sa forme définitive il devrait être parfait - vous établirez d'abord une ébauche et vous affinerez l'algorithme au fur et à mesure de vos progrès. Assurez-vous toujours que votre algorithme est complet avant de commencer à écrire des instructions.

Organigramme

Ainsi nous avons défini l'algorithme. Pourtant, avant de traduire cet algorithme en un programme BASIC, il est souhaitable d'en établir l'*organigramme.* En négligeant de le faire, vous risquez de vous trouver devant un programme qui ne fonctionne pas et dont la correction exigera bien des efforts. Un bon organigramme vous évitera des erreurs et des pertes de temps.



«Je suis l'organigramme. J'adore me rendre utile. A votre service.

Un *organigramme* est un diagramme qui «met à plat» et présente sous forme de tableau l'enchaînement des opérations à effectuer en vue de la solution recherchée. Ces opérations, on vient de le dire, constituent l'algorithme. L'organigramme n'est

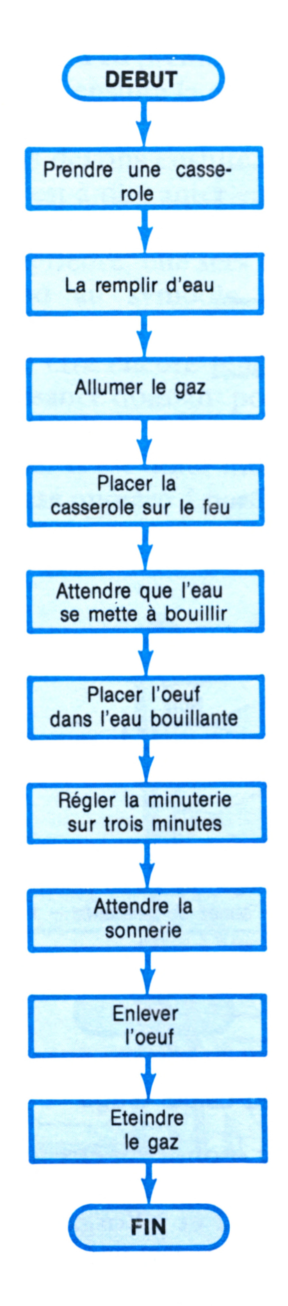


Figure 8.1 :
Préparation d'un œuf à la coque

donc qu'une représentation graphique de l'algorithme.

Plus tard, quand vous serez familiarisé avec l'organigramme, vous pourrez vous dispenser de définir l'algorithme et vous attaquer directement au tracé de l'organigramme.

La Figure 8.1 représente l'organigramme de la préparation d'un œuf à la coque.

Etudions maintenant un nouveau programme qui demande votre date de naissance ainsi que la date du jour et calcule ensuite votre âge. L'algorithme est évident. La Figure 8.2 représente l'organigramme correspondant.

Les symboles en forme de losange indiquent un test c'est-à-dire un choix. Assurez-vous donc que chaque choix comporte deux résultats possibles : «oui» ou «non», ceci pour faciliter la conversion de l'organigramme en un programme. Marquez les flèches correctement. Maintenant examinons la Figure 8.2 et vérifions que chaque flèche sortant d'un losange porte la mention «oui» ou «non» selon le résultat du test.

Il y a trois manières de tracer les flèches, comme le montre la Figure 8.4. Vous pouvez choisir celle qui vous convient. C'est-à-dire celle qui vous paraît améliorer la lisibilité de l'organigramme. La position des flèches

DEBUT

Quelle est la date du jour — 1

Quelle est la date de votre anniversaire — 2

(erreur)

Anniversaire > date du jour ? — 3

Oui

Non

Mois anniversaire < mois en cours ? — 4

Oui

Non

Age = année en cours − année de naissance — 5

Age = année en cours − année de naissance − 1 — 6

FIN

FIN

Figure 8.2 : Organigramme du calcul d'âge

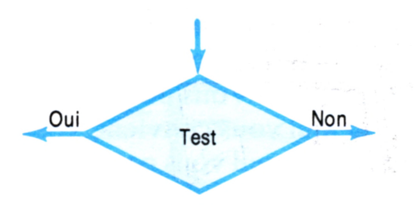


Figure 8.3 : Symbole de test

«oui» et «non» peut être inversée.

Revenons maintenant à la Figure 8.2 et à l'organigramme de calcul de l'âge et examinons l'algorithme correspondant.

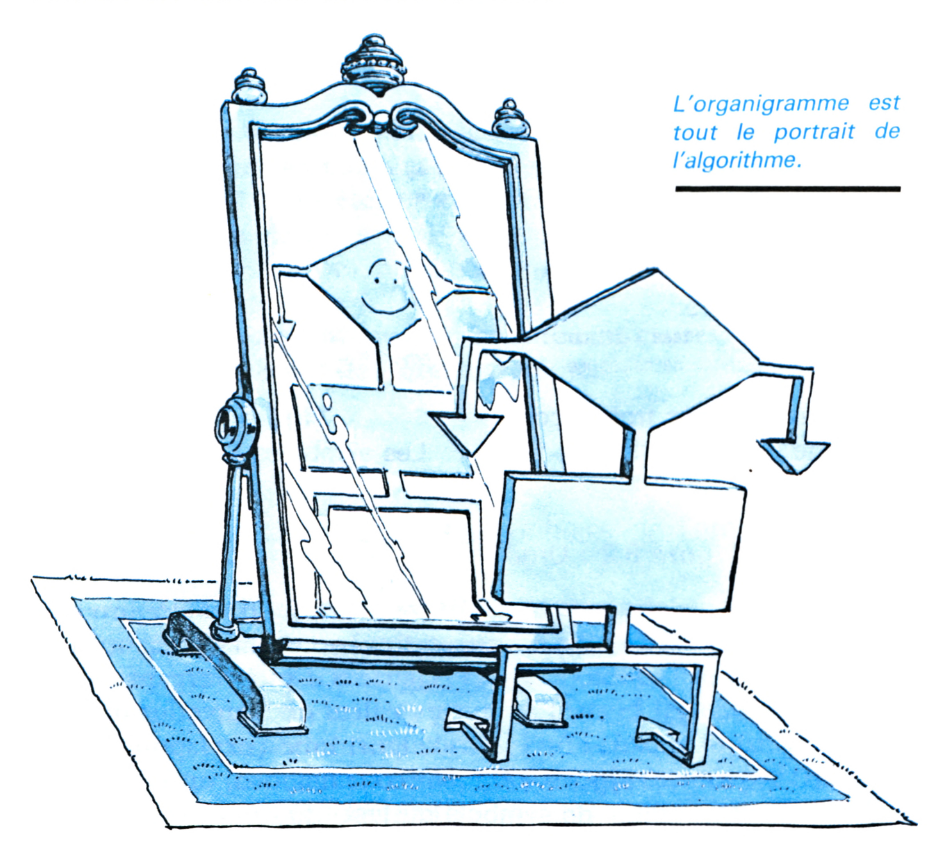
Tout d'abord, on y demande la date du jour. Cette étape

correspond au premier symbole de l'organigramme (numéro 1). On demande ensuite la date de naissance de l'utilisateur : la question correspond à la case 2 de l'organigramme.

Nous devons ensuite vérifier que la date de naissance est logique, c'est-à-dire antérieure à la date du jour. Si elle est postérieure, l'erreur sera décelée et il faudra recommencer le processus. Si elle est antérieure, elle sera considérée comme juste. Cette étape correspond au symbole en forme de losange (numéro 3) sur l'organigramme.

Pour être encore plus précis, nous pourrions refuser les dates de naissance donnant pour résultat 150 ans et plus, un résultat qui paraît absurde. Cependant, accepter de telles dates n'a pas d'effet nuisible. Il est donc inutile de les vérifier.

La case numéro 4 permet de déterminer si le mois de votre anniversaire est antérieur au mois en cours.



L'organigramme est tout le portrait de l'algorithme.

Oui Test Non

1

Oui Test

Non

2

Test Oui

Non

3

DEBUT

FIN

Figure 8.4 : Les trois façons de placer les flèches

Figure 8.6 : Les symboles DEBUT et FIN

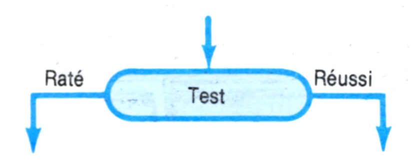


Figure 8.5 : Autre forme de symbole de test

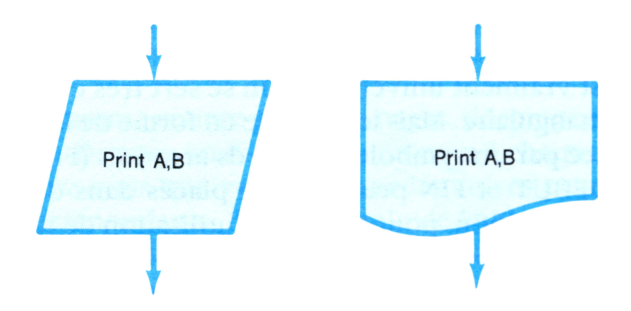


Figure 8.7 : Symboles pour l'instruction PRINT

S'il l'est, votre anniversaire a déjà eu lieu et votre âge peut être calculé (case 5) en faisant la différence entre l'année en cours et l'année de votre naissance. Par exemple, si vous êtes né en février 1946 et si nous sommes en mars 1983, votre âge est 1983 – 1946 = 37.

S'il ne l'est pas (case 6), votre âge se calcule en faisant la différence entre l'année en cours et l'année de votre naissance, moins 1. Par exemple, si vous êtes né en juin 1942 et que nous sommes en mars 1983, votre âge est : 1983 – 1942 – 1 = 40. Pour ne pas compliquer les choses, nous n'allons pas vérifier le jour. Nous le ferons plus tard.

Les étapes de l'algorithme sont maintenant claires. Examinons les différents symboles de l'organigramme.

Les symboles de l'organigramme

Dans un organigramme, les symboles rectangulaires sont utilisés pour les calculs et les opérations qui ne nécessitent pas de choix, tels qu'une entrée ou un affichage. Les symboles en forme de losange sont, nous l'avons vu, utilisés pour les tests et les choix. Ils doivent toujours comporter deux sorties. Enfin chaque organigramme doit comporter un début et une fin qui sont représentés par DEBUT (BEGIN ou START) et FIN (END)

Les symboles utilisés dans les organigrammes ne sont pas uniformément normalisés. Plusieurs normes ont été proposées mais aucune n'est vraiment universelle. On se sert très couramment du symbole rectangulaire. Mais le symbole en forme de losange est parfois remplacé par un symbole aux bords arrondis (Figure 8.5). Les symboles DEBUT et FIN peuvent être placés dans des petits cercles (Figure 8.6). Enfin, pour indiquer l'utilisation de tel ou tel périphérique on a le choix entre divers symboles. Une opération PRINT peut être symbolisée de deux façons (Figure 8.7).

En fait, les symboles ne sont qu'une convention. L'organigramme est simplement un moyen pratique de visualisation d'un algorithme. Vous pouvez, si vous le souhaitez, choisir vous-même vos symboles. Il est préférable, cependant, d'utiliser des symboles courants afin que d'autres personnes puissent se servir de vos programmes et que vous puissiez lire et suivre sans difficulté les organigrammes des autres programmeurs.

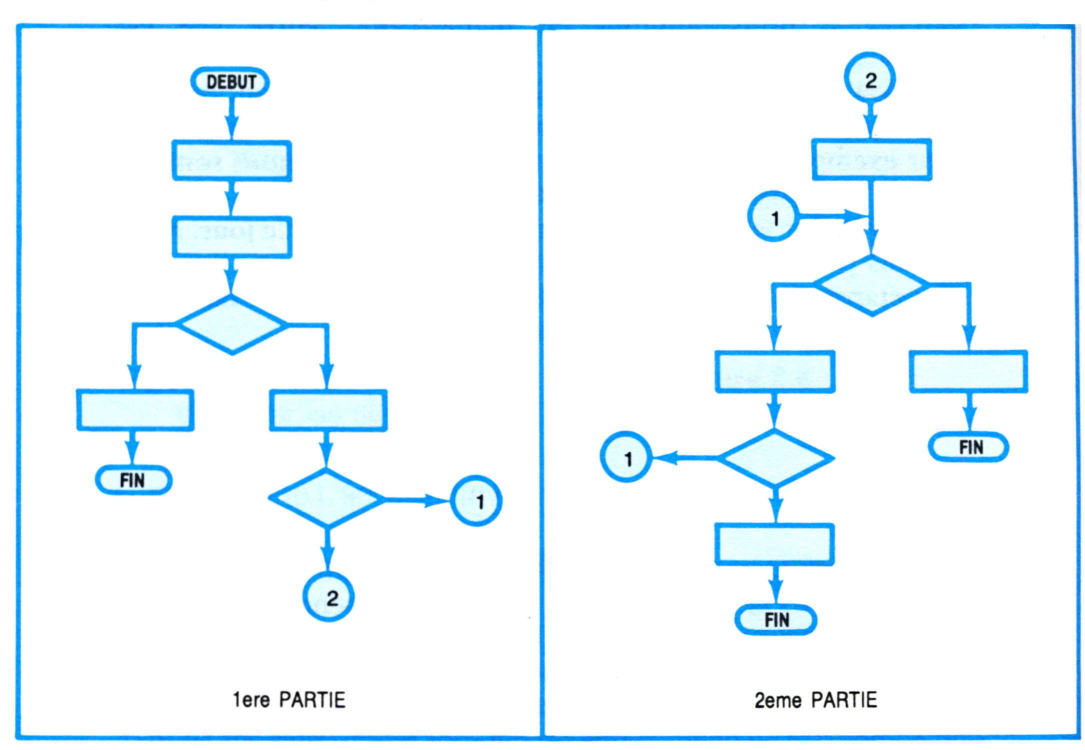


Figure 8.8 : Découpage de l'organigramme

Découpage de l'organigramme

Si un organigramme couvre plusieurs pages, décomposez-le en plusieurs parties. Donnez un numéro ou un nom à chaque flèche coupée, et assurez-vous que vous avez sur les autres pages les points d'entrée correspondants. Sur l'exemple de la Figure 8.8, on a utilisé des nombres pour relier les flèches.

Affinage de l'organigramme

Vous pouvez écrire, comme bon vous semble, les instructions placées dans les symboles de l'organigramme. Ce ne sont pas des instructions BASIC. Quand vous ferez le premier tracé de votre organigramme, vous inscrirez dans les symboles des instructions assez sommaires. Puis, vous affinerez peu à peu.

Par exemple : l'organigramme de la Figure 8.2, quand il détermine si votre anniversaire a déjà eu lieu entre le début de l'année et le mois en cours, vérifie le mois de votre naissance, mais ne vérifie pas le jour. Affinons cet organigramme. La partie correspondante de l'organigramme initial (ébauché) apparaît à la Figure 8.9. La Figure 8.10 représente le nouvel organigramme.

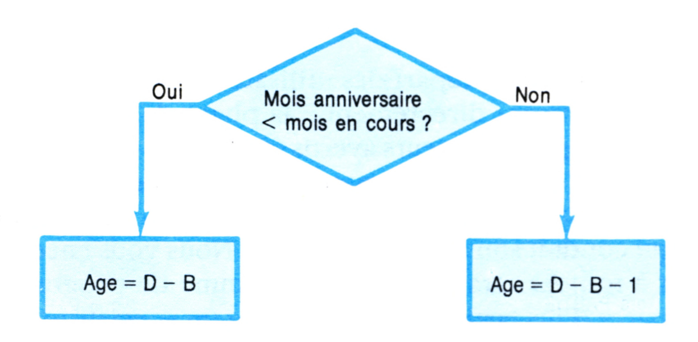


Figure 8.9 : Algorithme simple

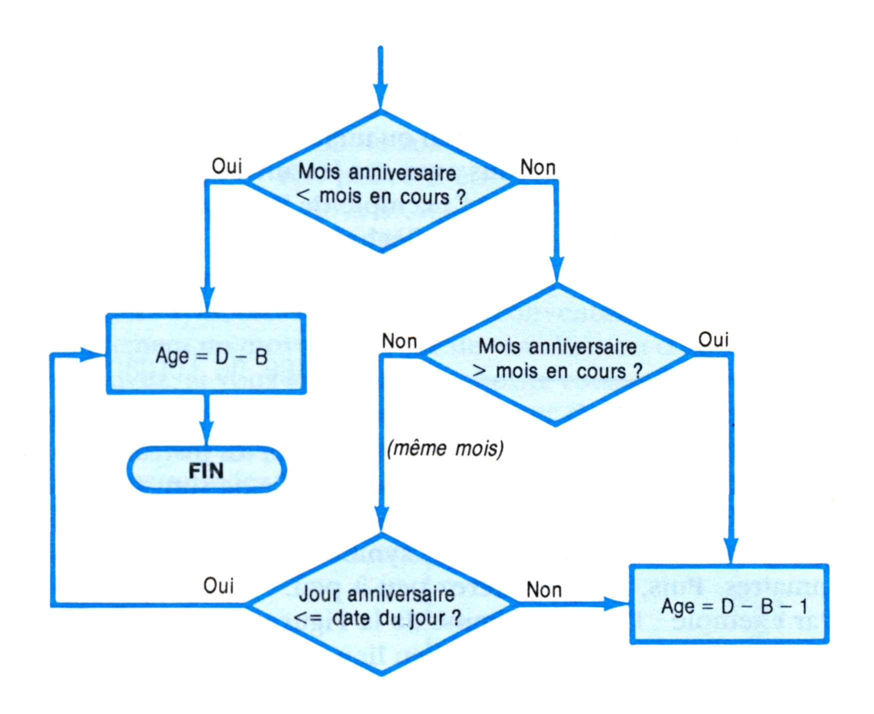


Figure 8.10 : Algorithme affiné

Dans la pratique, la plupart des utilisateurs ne rédigent pas d'algorithme ; ils passent directement à la phase de l'organigramme. Quelquefois, les programmeurs avertis (et d'autres qui le sont moins) sautent ce stade de l'organigramme et commencent à écrire directement le *programme* sur papier. C'est une façon de procéder dangereuse qui conduit souvent à des erreurs. Nous vous encourageons vivement à toujours tracer un organigramme du programme que vous préparez. Plus tard, quand vous serez un spécialiste de la programmation, vous pourrez vous dispenser de l'organigramme détaillé, et vous contenter d'une ébauche.

Une fois que vous avez écrit l'organigramme, testez-le à l'aide d'exemples réels. Assurez-vous que le résultat est correct ou du moins semble l'être. Cette méthode s'appelle le *test manuel* par opposition au test sur ordinateur.

Par exemple, revenons à l'organigramme de calcul de l'âge, à la Figure 8.2, et donnez comme indiqué la date du jour et votre date de naissance. Obtenez-vous votre âge exact ? Si oui, les choses se présentent bien. Sinon, il y a une erreur. Maintenant essayez à nouveau en prenant des valeurs différentes, en utilisant des dates de naissance antérieures et postérieures à la date du jour. Tout va bien ? Alors l'algorithme est probablement correct. Le test manuel est une méthode rapide pour s'assurer qu'il n'y a pas de faute évidente.

Ayant tracé un organigramme qui semble fonctionner, nous avons accompli toutes les tâches préliminaires à la réalisation du programme. Maintenant écrivons ce programme.

A sauter des étapes, vous aurez des problèmes.



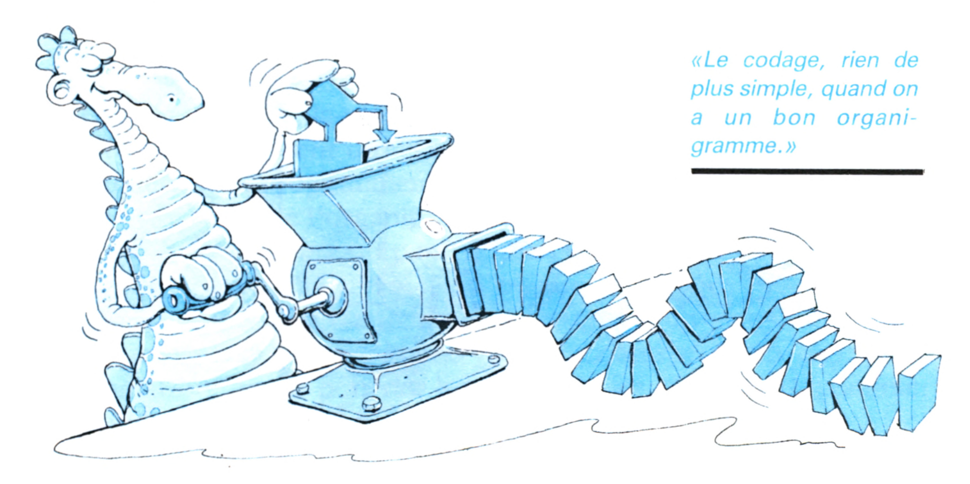
Le codage

L'opération qui consiste à écrire les instructions d'un programme s'appelle le *codage*. Le terme *programmation* englobe normalement toutes les phases de création d'un programme que nous venons d'étudier : définition de l'algorithme, organigramme, codage, mise au point et test. Le codage est la traduction du contenu des symboles de l'organigramme en instructions de programme énoncées en langage de programmation - dans le cas présent, en BASIC.

Le codage sera aisé si l'on a écrit un organigramme suffisamment détaillé. Le programmeur novice préparera l'organigramme de façon que chaque symbole ne contienne que la matière d'une ou de deux instructions BASIC. Le contenu de chaque symbole doit pouvoir être traduit en instructions BASIC simples. Plus tard, quand il aura acquis de l'expérience, il sera en mesure d'écrire des organigrammes plus «vagues» dont chaque symbole sera codé en plusieurs instructions BASIC.

Dans la création d'un programme, la phase de codage est souvent celle qui exige le moins de temps. Il faut plus de temps pour tester un programme que pour le coder. Si l'organigramme est bien fait, la phase de codage sera d'autant plus courte.

Quand vous codez un programme, efforcez-vous de le rendre clair et précis afin qu'il fonctionne rapidement et puisse être facilement testé ou modifié.



«Le codage, rien de plus simple, quand on a un bon organigramme.»

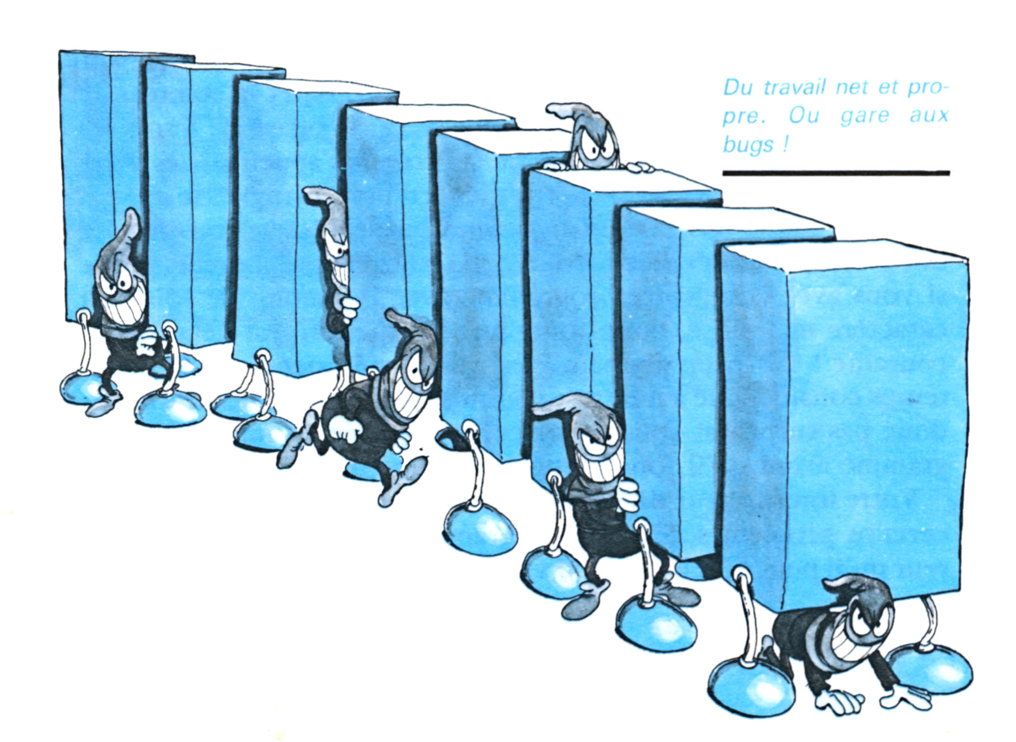
Un programme précis

Ecrivez les instructions de votre programme avec le plus grand soin : toute erreur de positionnement d'un signe de ponctuation ou d'un symbole fera probablement «avorter» le programme.

Un programme clair

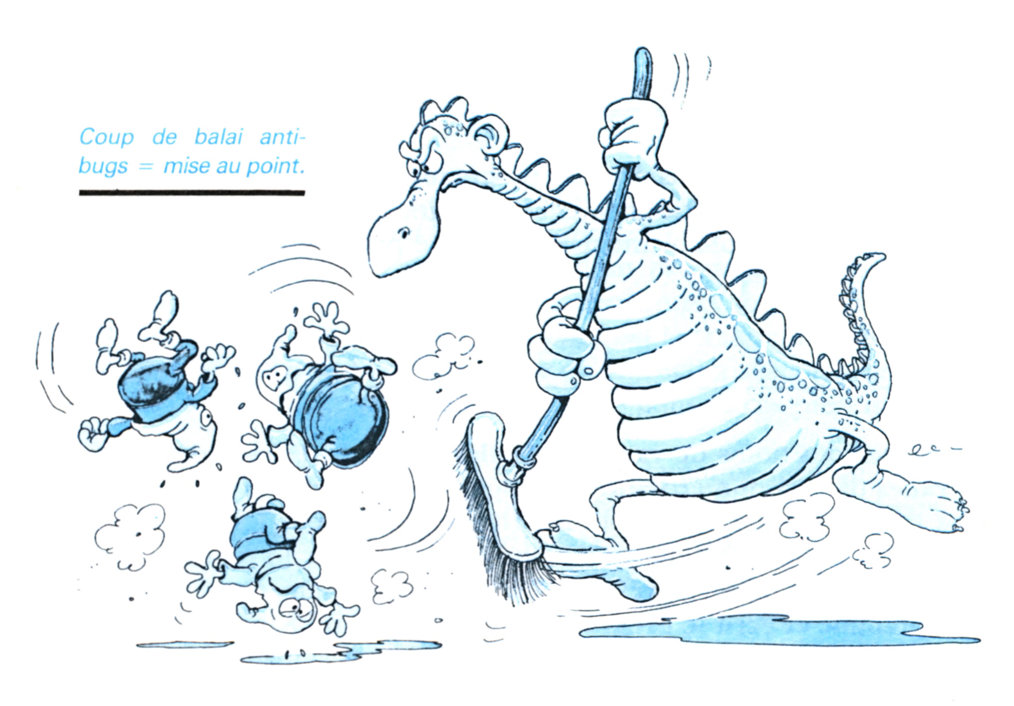
Utilisez des noms de variables faciles à retenir. Laissez des intervalles dans la numérotation des lignes pour le cas où vous auriez à insérer d'autres instructions. Utilisez largement les remarques (l'instruction REM) dans votre programme pour le rendre clair.

Vous avez donc, à présent, défini un algorithme, tracé un organigramme et écrit le programme correspondant. Sans doute vous attendez-vous à ce qu'il fonctionne. Mais dans la majorité des cas, *les programmes ne fonctionnent pas la première fois.* Il faudra faire plusieurs essais, voire une quantité d'expériences, avant que le programme fonctionne correctement, quelle que soit sa longueur. Et nous entrons dans la phase suivante : la mise au point.



Du travail net et propre. Ou gare aux bugs !

Coup de balai anti-bugs = mise au point.



Mise au point

Vous avez codé votre organigramme en un programme BASIC. Votre programme est encore sur papier. Vous devez maintenant l'introduire dans la mémoire de l'ordinateur, taper la commande RUN et vous assurer qu'il fonctionne. C'est la phase de *test* et *de mise au point.* Dans un programme, les erreurs s'appellent des *bugs*. L'opération qui consiste à corriger les erreurs (en anglais le *debugging*) est, en français, la *mise au point.* Chaque fois que vous décelez une erreur, vous devez la corriger, et relancer le programme. Même si vous avez écrit votre programme avec beaucoup de soin. Un caractère mal placé, voire toute une instruction, est hélas chose courante ! Les programmeurs les plus expérimentés passent un temps considérable à mettre au point leurs programmes. Ne soyez donc pas surpris si vous devez corriger plusieurs fois votre programme avant qu'il fonctionne correctement.

Votre interpréteur BASIC est là pour vous aider à diagnostiquer certains problèmes. Si votre programme comporte un type d'erreur qu'il peut déceler, quand vous aurez tapé la commande RUN,

l'exécution du programme s'arrêtera, et l'interpréteur vous établira un diagnostic du genre : ERROR IN LINE 84 (ERREUR DE SYNTAXE A LA LIGNE 84). Il vous aide à déceler la plupart des *erreurs de syntaxe* (comme l'emploi de symboles incorrects). Malheureusement, il ne vous aidera pas à déceler les erreurs les plus graves, à savoir les *erreurs logiques* ou *de conception.* Dans le domaine de la logique, vous êtes le seul responsable. Vous devez définir soigneusement vos organigrammes, vous devez valider en le vérifiant tout nombre fourni à un programme ou généré par celui-ci. Si votre programme contient une erreur logique, cette technique vous permet d'isoler la partie du programme contenant l'erreur.

Dans le cas d'un programme simple, l'interpréteur BASIC peut généralement déceler quelques erreurs typographiques. Vous les corrigerez et votre programme fonctionnera. Assurez-vous toutefois qu'il fonctionne correctement en le testant avec différentes valeurs et dans différents cas ; votre programme peut comporter des erreurs logiques.



L'interpréteur aide à détecter les erreurs de syntaxe.

Conseils pratiques

Voici un conseil pratique : insérez des instructions PRINT supplémentaires tout au long de votre programme afin de vérifier des valeurs clés. Cela vous aidera à déceler des valeurs étrangères et isoler les instructions qui en sont la cause. Voici un exemple d'instruction PRINT que vous pouvez insérer :

```
1235  PRINT"TEST VALEUR MOYENNE";MOYENNE
```

Une fois que le programme fonctionne, vous pouvez supprimer ces instructions PRINT supplémentaires. Cette technique s'appelle le *pistage* d'une variable.

Autre conseil pratique : chaque fois que votre programme s'arrête de lui-même ou par l'intermédiaire de l'interpréteur, utilisez le *mode d'exécution immédiat* pour vérifier la valeur des différentes variables. Par exemple, vous pouvez taper :

```
PRINT MOYENNE
```

Puis :

```
PRINT SOMME
```

pour vérifier les valeurs de ces deux variables.

A présent votre programme fonctionne. Vous pensez qu'il est correct, vous ne voulez plus le modifier. Supposons toutefois qu'une erreur y soit décelée par la suite, ou simplement que vous vouliez réutiliser le programme ou le prêter à quelqu'un. Vous allez le faire passer par une dernière phase : celle de la *documentation.*

Documentation

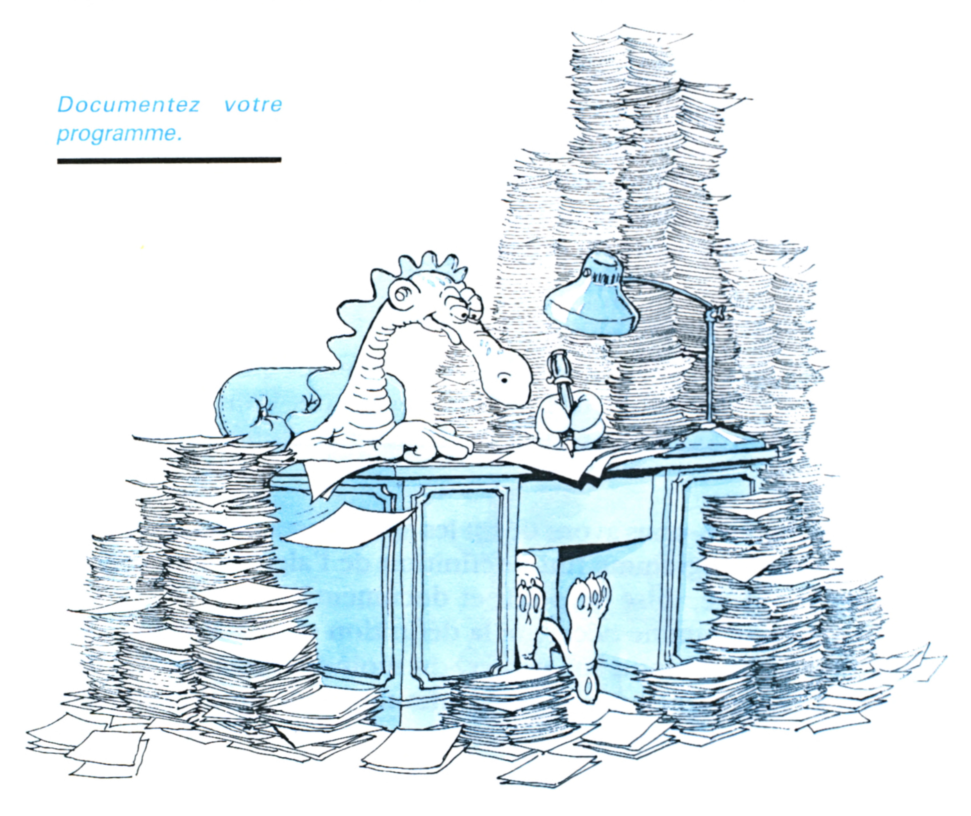
Vous venez juste de définir et de coder un programme. Vous êtes familiarisé avec son fonctionnement et avec chaque instruction.

Mais vous risquez de découvrir à vos dépens que l'on oublie très vite les objectifs, la finalité d'un programme. Si vous avez l'intention de réutiliser votre programme ou de corriger les erreurs décelées éventuellement par la suite, il est indispensable, nous l'avons déjà dit, de rendre le programme aussi clair que possible et d'établir une documentation (soit sur papier, soit sous forme de REMarques, à l'intérieur du programme) qui permettra de le relire avec le minimum d'efforts.

Nous vous avons déjà donné au Chapitre 5, des techniques de clarification d'un programme. Nous résumerons ici les règles qui faciliteront votre travail.

Adopter une présentation claire : séparer les parties essentielles par des lignes blanches et utiliser des alignements ou des décalages. Prendre des étiquettes ayant le même nombre de chiffres pour aligner verticalement les instructions du programme. Chaque fois qu'une instruction FOR...NEXT est utilisée, il est judicieux de décaler le bloc d'instructions inséré entre FOR et NEXT.

Documentez votre programme.



Utiliser largement les espaces pour clarifier les instructions complexes, en particulier celles contenant des expressions mathématiques. Placer des parenthèses pour mettre en évidence les résultats d'un calcul.

Expliciter les diverses opérations : insérer des instructions REM pour expliquer les formules, les tests, les identificateurs ou les conventions. Faire un bref compte rendu écrit de n'importe quelle méthode ou technique dont on se sert et qui n'est pas évidente ou qui n'est pas décrite par l'instruction PRINT à l'intérieur du programme.

Mettre de l'ordre dans les organigrammes : définir un organigramme ou un ensemble d'organigrammes ordonné correspondant exactement au programme. Durant le processus de mise au point, des modifications de dernière minute sont souvent apportées directement sur le programme. S'assurer qu'elles ont été reproduites sur l'organigramme d'origine.

Renuméroter les lignes : on a souvent besoin d'insérer des instructions supplémentaires durant le processus de correction du programme, c'est-à-dire durant la phase de mise au point. Dès que le programme fonctionne, renuméroter les lignes pour que toutes les étiquettes soient espacées de façon régulière, ce qui facilitera toute modification ultérieure du programme. Cette précaution n'est pas obligatoire.

Résumé

Dans ce chapitre, nous avons décrit les cinq phases nécessaires pour aboutir à un programme fini : définition de l'algorithme, organigramme, codage, mise au point et documentation.

Chaque programme nécessite la définition d'un algorithme qui doit être élaboré sinon sur papier, du moins mentalement. On a intérêt à l'esquisser en une série de formules ou de notes expliquant les étapes essentielles.

La phase suivante est la définition de l'organigramme qui décrit toute la suite des événements. Considérons-la comme la phase principale pour tout programme nécessitant plusieurs lignes. Répétons-le : plus la définition de votre organigramme est soignée, plus vous avez de chances que votre programme soit correct.

Puis, nous passons au codage : l'organigramme est traduit en instructions BASIC. La pratique va accélérer considérablement cette phase. En fait, la phase du codage devient très rapidement la phase la plus courte.

C'est ensuite la phase de test et de mise au point, indispensable et souvent la plus longue. Chaque programme doit être soigneusement contrôlé.

Enfin, une documentation soignée facilitera l'utilisation ultérieure du programme et permettra de lui faire subir, sans trop de risques d'erreurs, d'éventuelles modifications.

Exercices

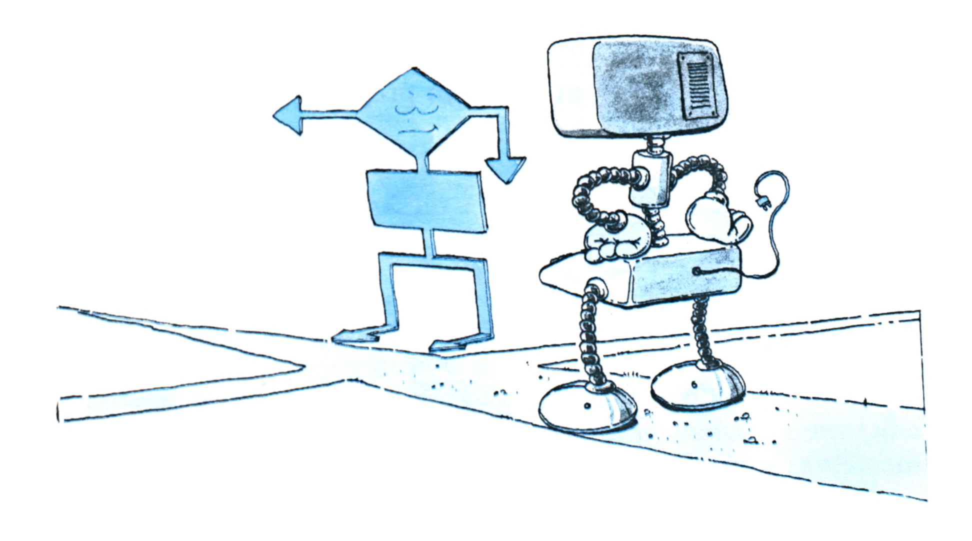
8-1 : Quelle est la différence entre codage et programmation ?

8-2 : Comment pister une variable ?

8-3 : Qu'est-ce qu'un organigramme ?

8-4 : Quels sont les avantages d'un programme clair ?





9

ETUDE DE CAS : CONVERSION METRIQUE

Nous allons maintenant mettre au point un programme complet et décrire successivement chaque étape. Voici le problème à résoudre : il s'agit de réaliser un programme qui convertit automatiquement un poids exprimé en onces en sa valeur en grammes. Ce programme doit soit convertir un nombre tapé au clavier soit imprimer une table de conversion de poids pour des nombres situés entre deux valeurs définies.

Définition de l'algorithme

La suite des phases par lesquelles nous allons passer pour résoudre ce problème est évidente : nous demanderons à l'utilisateur ce qu'il veut (une simple conversion ou une table des valeurs) et nous exécuterons la tâche demandée. Telle est l'ébauche de notre algorithme. Affinons-la.

Une once est égale à 28.35 grammes. La conversion des onces en grammes s'effectue donc selon la formule suivante :

Pgrammes = Ponces × 28.35

ou, en abrégé :

Pg = Poz × 28.35

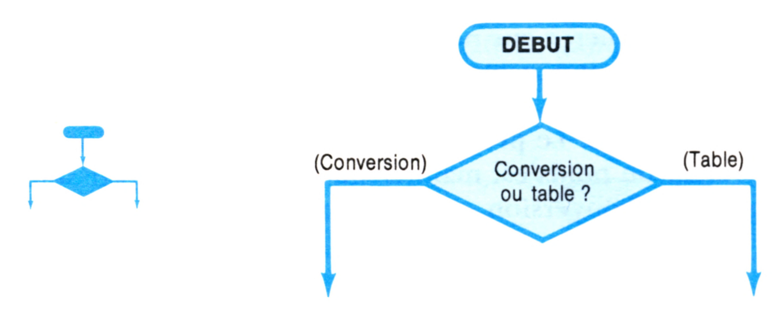
Voici l'algorithme de base :

- définir s'il s'agit d'une conversion ou d'une table
- s'il s'agit d'une conversion, demander le poids en onces
- convertir en grammes (en utilisant la formule ci-dessus) et afficher le résultat
- END
- s'il s'agit d'une table, demander le poids maximum en onces
- convertir en grammes et afficher les résultats jusqu'à la limite autorisée
- END.

Dans la pratique, il n'est pas nécessaire de rédiger l'algorithme puisque vous allez tracer l'organigramme.

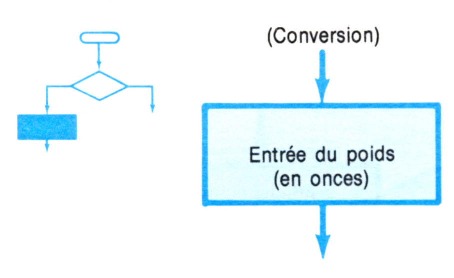
L'organigramme

Vous devez d'abord déterminer si l'utilisateur souhaite que le programme fasse seulement la conversion ou affiche une table des valeurs. Voici l'élément d'organigramme correspondant :

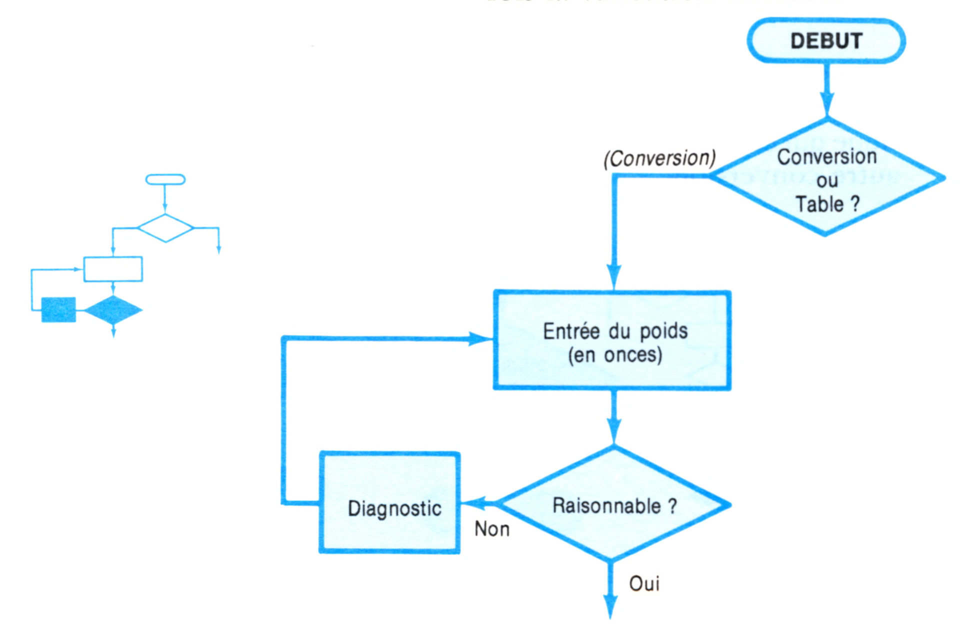


Voici un symbole de décision comportant deux sorties possibles (c'est-à-dire deux branches) : CONVERSION et TABLE.

Examinons tout d'abord CONVERSION : l'utilisateur désire convertir un seul poids exprimé en onces en grammes. Nous devons demander la valeur du poids. Voici l'entrée de l'organigramme correspondant :



Nous pourrons maintenant convertir cette valeur en grammes. Affinons tout de suite l'organigramme. Par précaution, nous allons valider la valeur fournie par l'utilisateur, en vérifiant si elle est acceptable ou non. Voici l'organigramme une fois la validation effectuée.

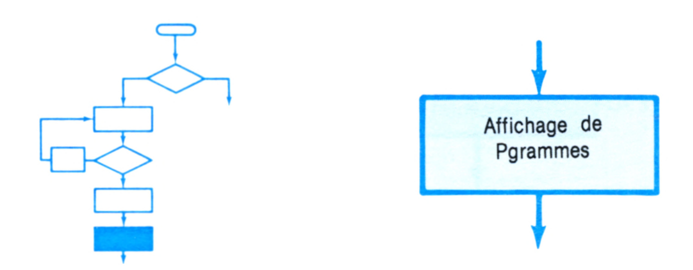


Vous remarquerez que nous avons ajouté un symbole pour vérifier si l'entrée est acceptable. Si oui, nous continuons. Sinon un diagnostic du genre : ENTREE INACCEPTABLE, ESSAYEZ A NOUVEAU est affiché. Le programme demande une autre valeur.

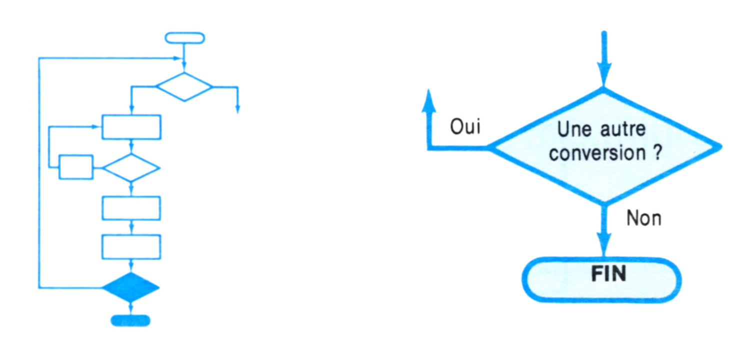
La valeur du poids est maintenant validée. Convertissons-la en grammes. La conversion se fait de la façon suivante :



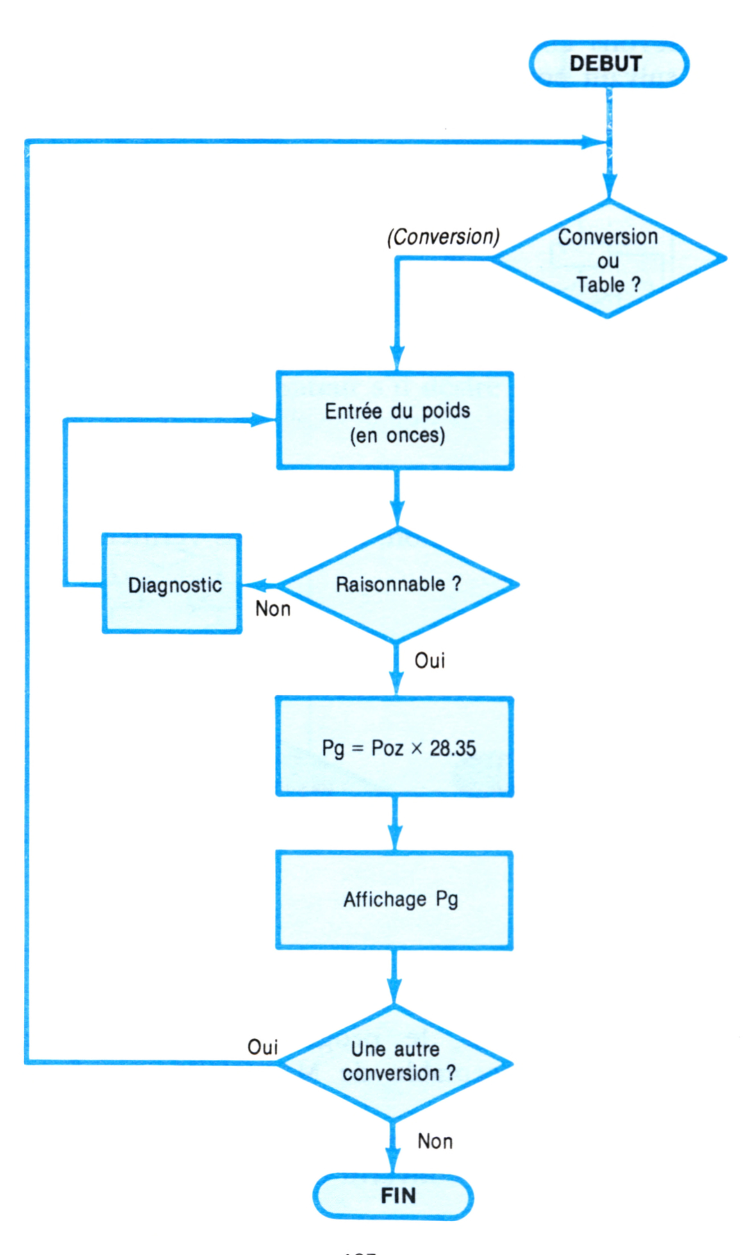
Nous allons maintenant afficher les résultats.



La conversion est effectuée. Nous pourrions terminer ici cette partie de l'organigramme. Cependant, ajoutons une fonction pratique qui consiste à demander à l'utilisateur s'il désire effectuer une autre conversion.

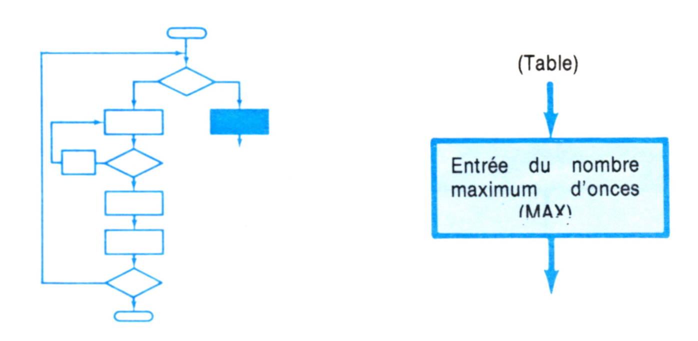


Le symbole DEBUT situé sur la flèche gauche indique que cette flèche sera reliée au début de l'organigramme. Notre organigramme prend alors la configuration suivante :

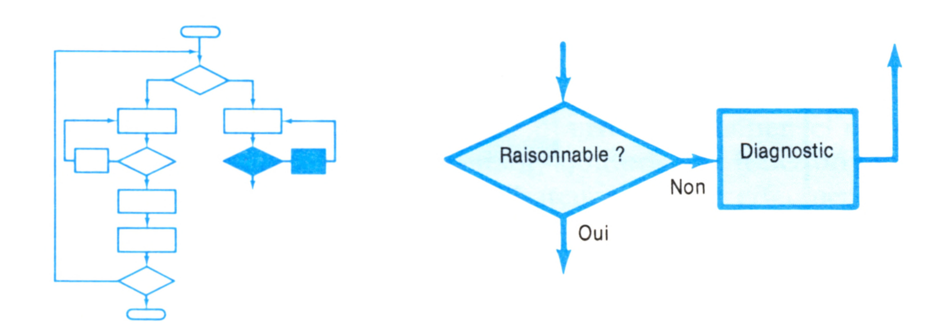


Revenons à notre premier symbole de décision et voyons ce qui se passe quand l'utilisateur désire afficher une table des valeurs. Le symbole en haut de l'organigramme correspond au choix de la TABLE.

Nous devons connaître la valeur maximum à convertir. Le symbole suivant permet de le faire :

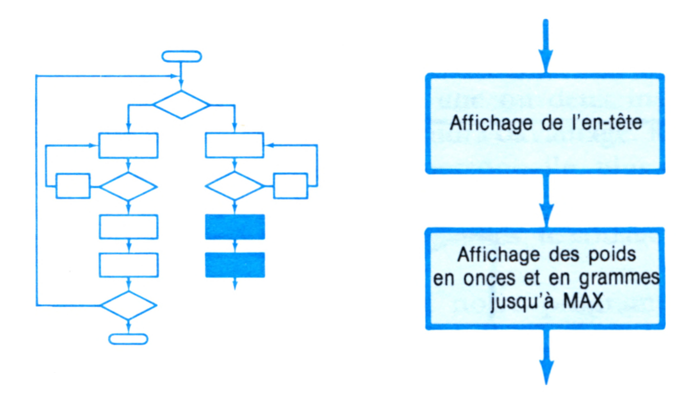


De nouveau, par précaution, nous allons vérifier si le nombre est acceptable ou non :

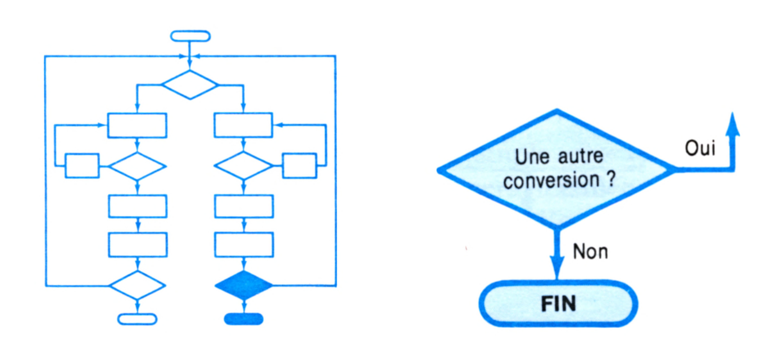


Comme précédemment, le programme ne continuera que lorsqu'une valeur acceptable de MAXIMUM sera fournie par l'utilisateur.

Une fois fournie une valeur acceptable, nous pouvons continuer et afficher une table qui convertit des onces en grammes jusqu'à la limite désirée :



Enfin, ajoutons la même fonction que ci-dessus, à savoir une fonction qui demande à l'utilisateur s'il désire effectuer une conversion supplémentaire :



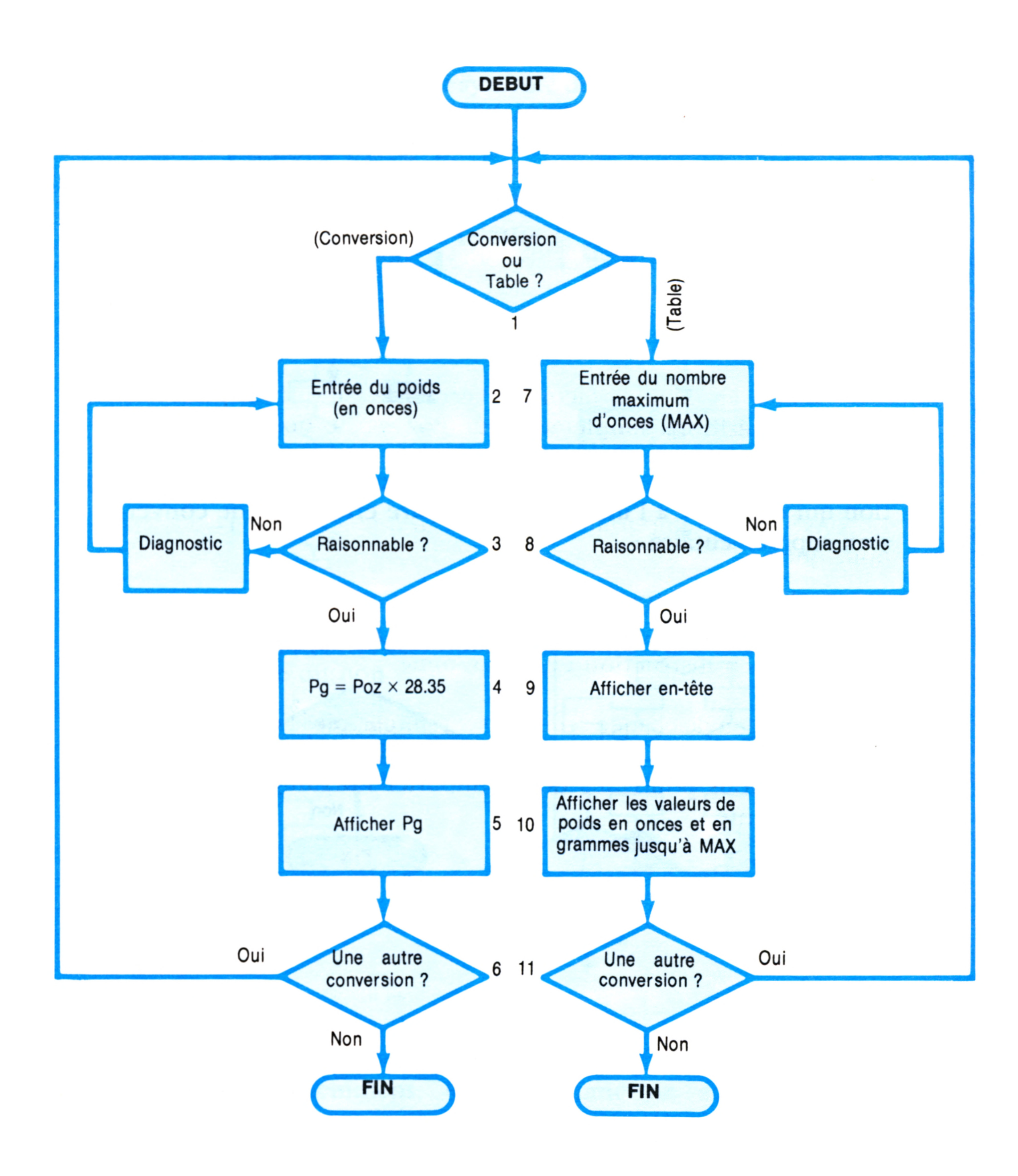


Figure 9.1 :

Organigramme de la conversion des mesures de poids

La Figure 9.1 représente l'organigramme complet. Si la suite des étapes est clairement établie, le contenu des symboles est quelque peu vague. Certains seront codés en une ou deux instructions BASIC, alors que pour d'autres, il en faudra davantage. Rappelons que vous pouvez choisir votre façon personnelle, plus ou moins explicite, d'écrire les symboles. Si les symboles sont simples, l'organigramme sera plus clair. S'ils sont détaillés, le codage sera plus facile.

Pour expliciter l'organigramme de notre programme, nous pourrions :

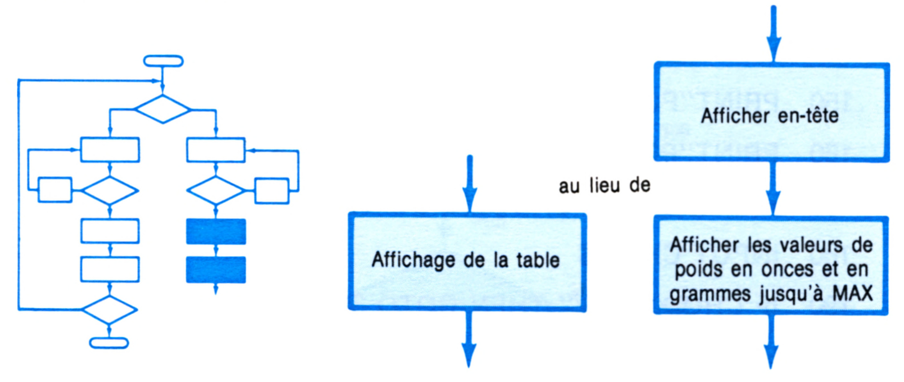
— Définir en détail la façon de vérifier la validité du poids (ici nous vérifions simplement que Poz est un nombre positif).

— Etre plus explicite quant au diagnostic et au dialogue.

Le conseil le plus important est : *restez simple.* Faites-en juste assez pour que :

1. La liste des étapes soit correcte et complète.
2. Chaque symbole soit compréhensible et facilement transformable en instruction de programme.

Plus le contenu de chaque symbole sera simple, plus l'organigramme sera clair. Par contre, plus chaque étape sera détaillée, plus le codage sera aisé. Nous pourrions donc simplifier l'organigramme de la façon suivante :



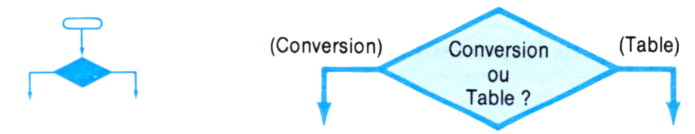
Les deux options sont correctes. A vous de décider ce qui vous convient le mieux. Pour faciliter le codage ou pour essayer une solu-

tion différente, il est toujours possible de réécrire le contenu d'un symbole ou une partie plus importante de l'organigramme sur une feuille séparée. Maintenant que l'organigramme est prêt, testons-le manuellement en employant des nombres pour vérifier son fonctionnement.

Le codage

Nous allons maintenant écrire un programme qui correspond à notre organigramme. Codons maintenant chaque symbole de l'organigramme en instructions BASIC correspondantes.

Voici le symbole 1 de l'organigramme.



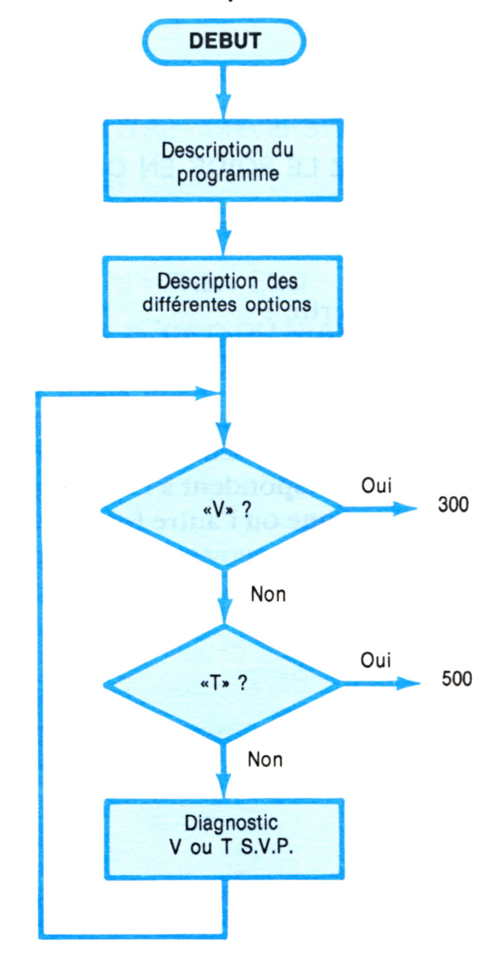
Voici les instructions correspondantes :

```
100  REM **CONVERSION ONCES-GRAMMES**
110  REM CE PROGRAMME CONVERTIT DIRECTEMENT
120  REM OU IMPRIME UNE TABLE DES VALEURS
130  REM DEFINIR LE MODE DE CONVERSION:DIRECTE OU
     TABLE
140  PRINT"JE CONVERTIS DES ONCES EN GRAMMES"
150  PRINT"POUR CONVERTIR TAPEZ V"
160  PRINT"POUR IMPRIMER UNE TABLE  TAPEZ T"
170  PRINT"VOTRE CHOIX (V OU T)";
180  INPUT CHOIX$
190  IF CHOIX$="V" THEN GOTO 300
200  REM ETIQUETTE 300 CORRESPOND A LA CONVERSION
     DE LA VALEUR
```

```
210  IF CHOIX$="T" THEN GOTO 500
230  REM SI LE CARACTERE EST DIFFERENT DE V OU T:
     ENTREE INCORRECTE
240  PRINT"V OU T S.V.P"
250  GOTO 170
```

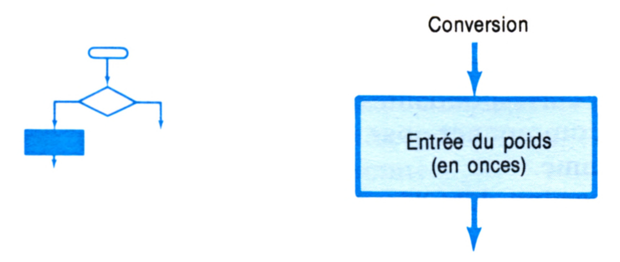
Une fois que vous aurez acquis plus d'expérience, vous serez en mesure de passer directement du symbole 1 de l'organigramme aux instructions correspondantes. Toutefois, au début, il sera nécessaire de commencer par écrire la version détaillée de l'organigramme.

Voici la version détaillée du symbole 1 :



Examinez cette version détaillée et remarquez la correspondance entre l'organigramme et les instructions BASIC. De plus, notez que nous avons introduit un *test de validité* pour la variable CHOIX$. Nous n'allons pas simplement décider que l'utilisateur va coopérer et taper «V» ou «T». Nous nous donnons la possibilité de vérifier la validité de l'entrée. Vérifiez. **Répétons-le :** chaque fois que vous demandez une entrée, vous devez la valider.

La suite est plus simple. Convertissons le symbole 2 :



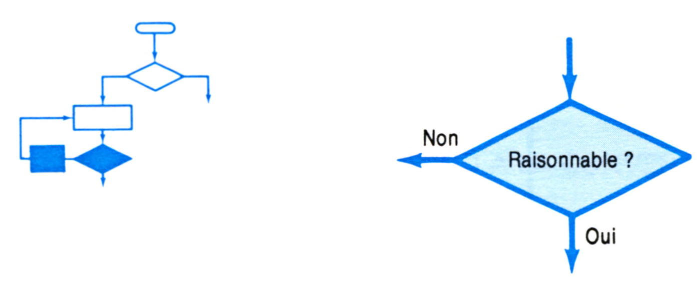
Les instructions correspondantes sont :

```
300 REM ** CONVERSION DE LA VALEUR **
310 PRINT"TAPEZ LE POIDS EN ONCES.. ";
320 INPUT POZ
```

Nous pourrions aussi écrire :

```
310 INPUT"TAPEZ LE POIDS EN ONCES....";POZ
```

Nous avons décidé ici de séparer les instructions PRINT et INPUT pour que vous vous rendiez mieux compte de la façon dont les lignes du programme correspondent à l'organigramme. Vous pouvez toutefois employer l'une ou l'autre forme. Voyons maintenant le symbole 3 :



Voici les instructions équivalentes :

```
330 IF POZ<0 THEN 310
340 REM LE POIDS DOIT ETRE POSITIF
350 REM NOUS POURRIONS AJOUTER UN MESSAGE
    EXPLICITE
```

L'équivalent du symbole 4 est :

```
360 PG=POZ * 28.35
```

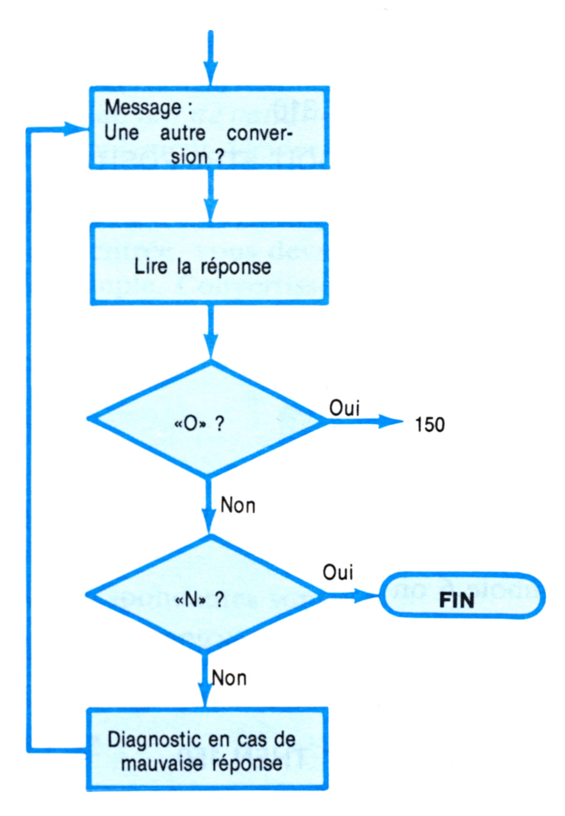
et celui du symbole 5 :

```
370 PRINT POZ;"ONCES =";PG;"GRAMMES"
```

Pour le symbole 6 on a :

```
410 PRINT"UNE AUTRE CONVERSION? (O=OUI, N=NON)"
420 INPUT AUTRE$
430 IF AUTRE$="O" THEN 150
440 IF AUTRE$="N" THEN END
450 REM ENTREE < > DE O OU N:AVERTIR L'UTILISATEUR
460 PRINT"O OU N S.V.P"
470 GOTO 380
```

Si cette partie du programme ne vous semble pas claire, voici l'organigramme équivalent :



Voyons maintenant la partie droite de l'organigramme de la Figure 9.1. Voici l'équivalent du symbole 7 :

```
520  PRINT"JE VAIS AFFICHER UNE TABLE"
530  PRINT"DE CONVERSION D'ONCES EN GRAMMES"
540  PRINT"NOMBRE D'ONCES ( > =1)"
550  INPUT MAXIMUM
```

Et pour le symbole 8 nous avons :

```
560  REM MAXIMUM SUPERIEUR OU EGAL A 1
570  IF MAXIMUM < 1 THEN GOTO 540
```

Pour être plus clair, sautons une ligne à l'affichage avant d'imprimer la table :

```
580  PRINT
```

Répétons-le : cette instruction PRINT vide affiche une ligne blanche. Maintenant voici l'équivalent du symbole 9 :

```
590  PRINT"ONCES","GRAMMES"
```

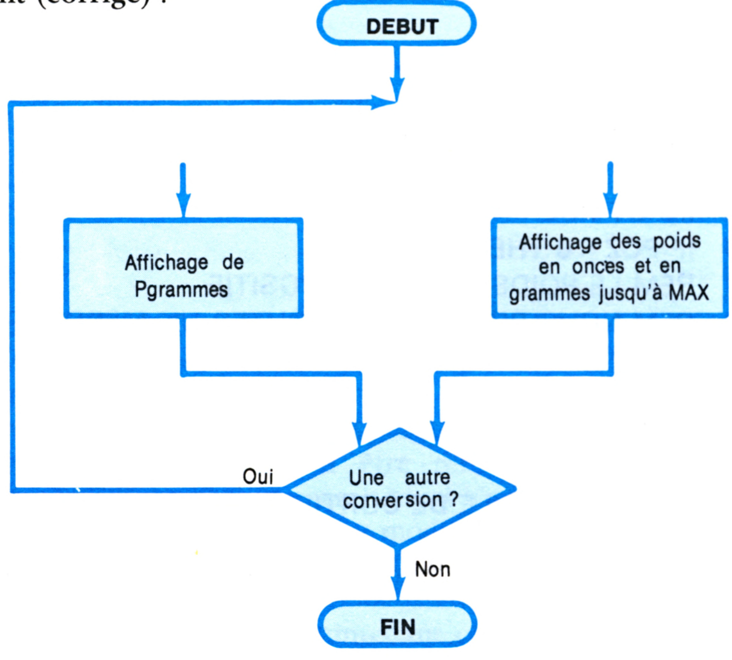
Vous remarquerez que nous utilisons une virgule plutôt qu'un point-virgule pour obtenir une présentation plus agréable des colonnes. L'équivalent du symbole 10 est :

```
600  I=1
610  PRINT I,I * 28.35
620  I=I+1
630  IF I< =MAXIMUM GOTO 610
```

Enfin voici le symbole 11 :

```
650  GOTO 380
```

L'instruction du symbole 11 est identique à celle du symbole 6 : nous pouvons donc simplifier notre programme en sautant (GOTO) aux instructions du symbole 6. Voici l'organigramme correspondant (corrigé) :



Vous trouverez ci-dessous le programme complet :

```
100  REM **CONVERSION ONCES-GRAMMES**
110  REM CE PROGRAMME CONVERTIT DIRECTEMENT
120  REM OU IMPRIME UNE TABLE DES VALEURS
130  REM DEFINIR LE MODE DE CONVERSION:DIRECTE OU
     TABLE
140  PRINT"JE CONVERTIS DES ONCES EN GRAMMES"
150  PRINT"POUR CONVERTIR TAPEZ V"
160  PRINT"POUR IMPRIMER UNE TABLE TAPEZ T"
170  PRINT"VOTRE CHOIX (V OU T)";
180  INPUT CHOIX$
190  IF CHOIX$="V" THEN GOTO 300
200  REM ETIQUETTE 300 CORRESPOND A LA CONVERSION
     DE LA VALEUR
210  IF CHOIX$="T" THEN GOTO 500
230  REM SI LE CARACTERE EST DIFFERENT DE V OU T:
     ENTREE INCORRECTE
240  PRINT"V OU T S.V.P"
250  GOTO 170
260  REM
270  REM
300  REM ** CONVERSION DE LA VALEUR **
310  PRINT"TAPEZ LE POIDS EN ONCES...";
320  INPUT POZ
330  IF POZ<0 THEN 310
340  REM LE POIDS DOIT ETRE POSITIF
350  REM NOUS POURRIONS AJOUTER UN MESSAGE
     EXPLICITE
360  PG=POZ * 28.35
370  PRINT POZ;"ONCES=";PG;"GRAMMES"
380  REM
390  REM ** MODULE DE SORTIE **
400  PRINT
410  PRINT"UNE AUTRE CONVERSION? (O=OUI, N=NON) "
420  INPUT AUTRE$
```

```
430 IF AUTRE$="O" THEN 150
440 IF AUTRE$="N" THEN END
450 REM ENTREE < > DE O OU N:AVERTIR L'UTILISATEUR
460 PRINT"O OU N S.V.P"
470 GOTO 380
480 REM
490 REM
500 REM ** TABLE DE CONVERSION **
510 REM DEMANDE DE LA LIMITE SUPERIEURE
520 PRINT"JE VAIS AFFICHER UNE TABLE"
530 PRINT"DE CONVERSION D'ONCES EN GRAMMES"
540 PRINT"NOMBRE D'ONCES ( > =1)"
550 INPUT MAXIMUM
560 REM MAXIMUM SUPERIEUR OU EGAL A 1
570 IF MAXIMUM<1 THEN GOTO 540
580 PRINT
590 PRINT"ONCES","GRAMMES"
600 I=1
610 PRINT I,I * 28.35
620 I=I+1
630 IF I< =MAXIMUM GOTO 610
640 REM I EST MAINTENANT>MAXIMUM.FIN DE TABLE
650 GOTO 380
660 END
```

Quelques modifications ont été apportées. Par exemple, plusieurs REM ont été ajoutées (voir instructions 260, 270, 300, 380, 390, 480, 490, 500).

D'autres modifications sont possibles. Par exemple, les instructions 600 à 630 peuvent être remplacées par l'instruction FOR...NEXT. La lisibilité serait ainsi légèrement améliorée.

Mais souvenez-vous que toute modification d'un programme peut être source d'erreur. **Un conseil :** ne le modifiez que si cela vous paraît vraiment utile.

Notre programme est maintenant complet. Essayons-le.

Lançons ce programme. Nous obtiendrons par exemple :

```
RUN
JE CONVERTIS DES ONCES EN GRAMMES
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V ou T)? V
TAPEZ LE POIDS EN ONCES...? 3
3 ONCES= 85.05 GRAMMES
```

Ou bien :

```
UNE AUTRE CONVERSION? (O=OUI, N=NON)
? O
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V OU T)? T
JE VAIS AFFICHER UNE TABLE
DE CONVERSION D'ONCES EN GRAMMES
NOMBRE D'ONCES (>=1)
? 4
ONCES    GRAMMES
1        28.35
2        56.7
3        85.05
4        113.4
```

Notre programme semble fonctionner aussi bien pour une seule conversion que pour une table de valeurs.
Essayons de le contrecarrer :

```
UNE AUTRE CONVERSION? (O=OUI, N=NON)
? O
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V ou T)? D
V OU T S.V.P
VOTRE CHOIX (V OU T)? V
TAPEZ LE POIDS EN ONCES...? 7
7 ONCES=  198.45 GRAMMES
```

Recommençons plusieurs fois :

```
UNE AUTRE CONVERSION? (O=OUI, N=NON)
? O
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V ou T)? V
TAPEZ LE POIDS EN ONCES....? 85
85 ONCES = 2409.75 GRAMMES

UNE AUTRE CONVERSION? (O=OUI, N=NON)
? O
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V OU T) V
TAPEZ LE POIDS EN ONCES....? 2.5
2.5 ONCES = 70.875 GRAMMES
```

Essayons à nouveau de le contrecarrer :

```
UNE AUTRE CONVERSION? (O=OUI, N=NON)
? O
POUR CONVERTIR TAPEZ V
POUR IMPRIMER UNE TABLE TAPEZ T
VOTRE CHOIX (V OU T)? V
TAPEZ LE POIDS EN ONCES...? -5
TAPEZ LE POIDS EN ONCES...?
```

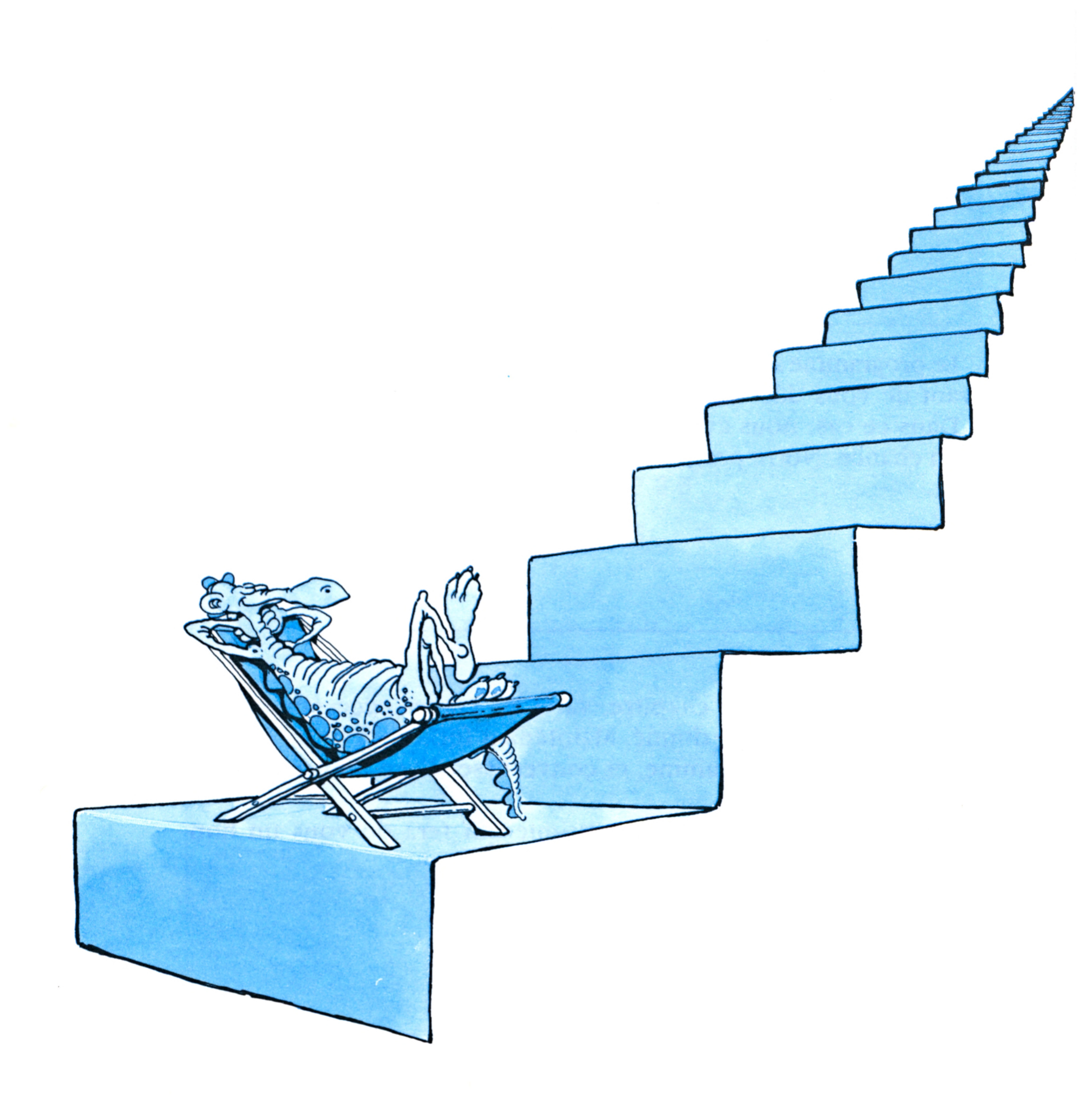
Le programme semble fonctionner. Mais faites de nombreux essais avant de vous déclarer complètement satisfait.

Dans ce cas, nous avons travaillé avec soin - et nous avons eu de la chance. Notre programme a bien fonctionné dès la première fois.

Résumé

Nous avons, dans ce chapitre, suivi les étapes nécessaires pour résoudre un problème donné. Maintenant fermez ce livre, écrivez votre propre organigramme, et convertissez-le en un programme qui fonctionne.

N'oubliez pas que c'est la pratique qui fera de vous un bon programmeur.



10

L'ETAPE SUIVANTE

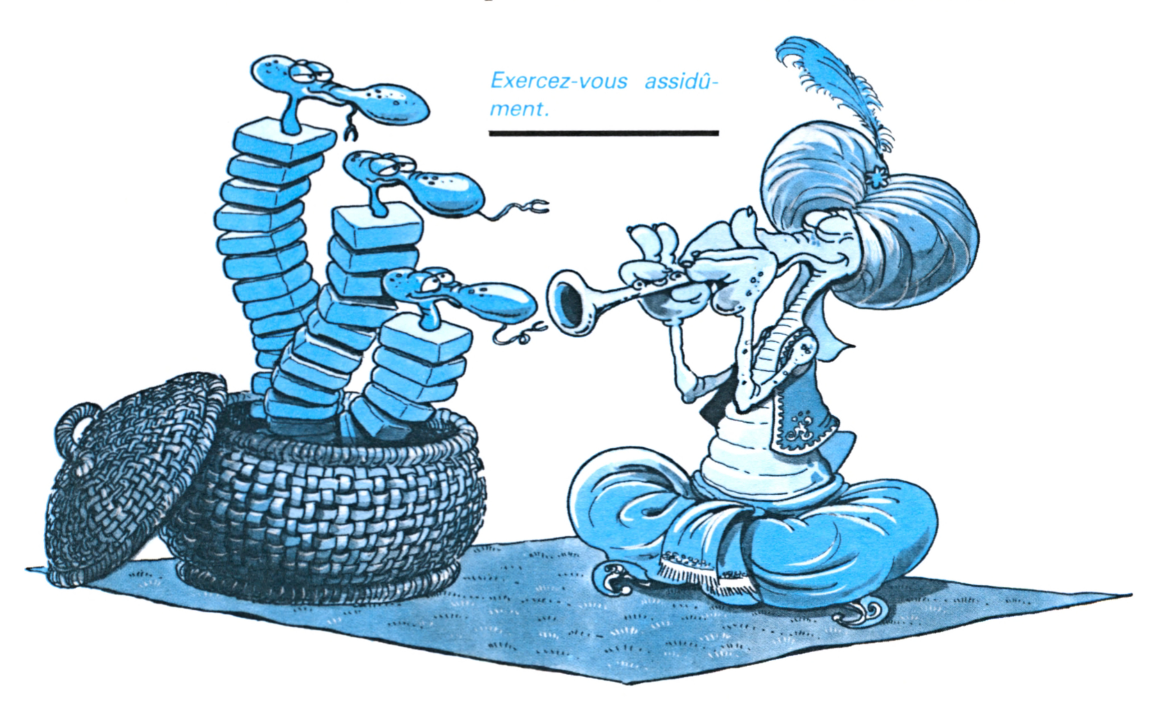
Vous savez maintenant écrire vos propres programmes BASIC. Dans ce chapitre, nous verrons de quelle façon vous pouvez améliorer vos compétences en matière de programmation. Nous allons revoir ce que vous pouvez faire avec le BASIC, et ensuite nous décrirons les méthodes et les techniques complémentaires qui vous permettront d'écrire plus aisément des programmes complexes.

Ce que vous pouvez faire en BASIC

Vous pouvez écrire un programme BASIC pour automatiser la plupart des tâches, à moins qu'elles n'exigent des calculs mathématiques très précis, des prises de décision complexes ou une réponse très rapide (tel le contrôle de processus en temps réel). Vous constaterez que le BASIC est très approprié aux applications simples comme le traitement des données, les listes de diffusion et les calculs financiers courants. Le BASIC de l'Amstrad se prête aussi aux graphiques et aux jeux.

Le BASIC a d'autres champs d'application tels que l'enseignement assisté par ordinateur (E.A.O.), la gestion de fichiers personnels et commerciaux. Il se prête aux calculs mathématiques et techniques (mais avec une précision limitée) et à bien d'autres utilisations. Les applications sont en général limitées par nos propres capacités en programmation.

Vous devrez maintenant être capable d'écrire une large variété de programmes BASIC. Cependant, au fur et à mesure de vos progrès, vous éprouverez le besoin d'accroître vos compétences et d'utiliser au maximum les possibilités de vos instruments de travail.



Exercez-vous assidûment.

Accroître vos compétences

Il vous faut :

1. Acquérir plus de pratique.
2. Inventorier toutes les ressources du BASIC de l'Amstrad.
3. Etudier des techniques complémentaires de programmation.

Plus de pratique

Comme nous l'avons dit et redit, en programmation la pratique est indispensable. Vous devez écrire le plus possible de programmes et les faire fonctionner en suivant, afin de prendre de bonnes habitudes, toutes les recommandations que l'on vous a données dans ce livre. Si vos programmes, après quelques essais, fonctionnent parfaitement, vous êtes sur la bonne voie pour devenir un programmeur efficace. Dans le cas contraire, remettez en question vos méthodes de travail, faites de nouveaux exercices. Revoyez certaines parties de ce livre.

Ce manuel vous a aidé à faire vos premiers pas, mais souvenez-vous que rien ne peut remplacer l'expérience.

Travaillez, prenez de la peine...



Caractéristiques de certains BASIC

Le BASIC de l'Amstrad offre des possibilités spécifiques qui comprennent des commandes, des instructions, des possibilités de commandes abrégées, les extensions au «BASIC standard» (telles que les graphiques et le son) et un environnement fonctionnel (commandes de sauvegarde sur disque ou cassette, fichiers, programme d'édition, et bien d'autres choses). Vous pouvez augmenter vos capacités en programmation en apprenant ces fonctions et méthodes supplémentaires. Dans une dernière partie, nous décrirons les instructions BASIC supplémentaires dont dispose le BASIC de l'Amstrad. Les techniques telles que les graphiques, le son et les fichiers sont particulières à votre ordinateur.

Techniques supplémentaires

Dès que vous aurez mis au point des programmes plus longs (c'est-à-dire de plus d'une page), vous éprouverez le besoin de connaître les techniques employées pour la résolution des problèmes courants, tels que le classement d'articles, leur tri, le formatage des données, les fichiers et la création de structures de données. Ces sujets sont en général traités dans les ouvrages de programmation.

Plus de BASIC

Le BASIC de l'Amstrad offre un ensemble «normalisé» de techniques auxquelles s'ajoutent de nombreuses extensions particulières. Ces facilités courantes ont été passées en revue. Il faut y ajouter six types d'instructions dont nous allons vous donner un bref aperçu afin que vous puissiez décider si vous voulez les étudier, soit seul, soit dans un livre plus spécialisé. Ce sont :

1. Les Fonctions : les fonctions sont des expressions soit *intégrées* soit *définies par l'utilisateur* qui opèrent sur une variable donnée et exécutent un calcul ou une opération particulière. Le BASIC de l'Amstrad offre des fonctions intégrées (telles que ABS, COS, EXP, RND, SGN, SIN, SQR, INT et TAN). Ces fonctions exécutent automatiquement des tâches courantes. L'utilisateur peut définir des tâches supplémentaires.

Etudiez, étudiez encore.



Examinons quelques-unes des fonctions intégrées types :

— La fonction INT calcule la partie entière d'un nombre décimal en éliminant la partie fractionnaire du nombre. Par exemple, INT (1.234) donne la valeur 1.
— De la même façon, ABS calcule la valeur absolue. Par exemple, ABS (−5) est 5.
— La fonction SQR calcule la racine carrée d'un nombre. Par exemple, SQR(4) donne la valeur 2.

L'utilisateur peut aussi définir des fonctions. Une fonction définie par l'utilisateur est essentiellement une formule qu'il a écrite et à laquelle il a donné un nom, qui opère sur une variable, et qui peut être utilisée dans le programme autant de fois qu'il le désire. Ensuite chaque fois que le nom de la fonction est utilisé, la formule est calculée avec la valeur courante de la variable. Il s'agit là d'une notation abrégée très commode. Voici un exemple de fonction définie par l'utilisateur qui calcule 2 % de X :

```
10 DEF FNA(X)=X * 2/100
```

Et voici une manière de l'utiliser :

```
20 PRINT FNA(50)
```

On obtient la valeur 1.

Examinons ces instructions de plus près. FN signifie fonction. FNA désigne la fonction A. C'est le nom de la fonction. X est la variable «fantôme» : X n'a pas de valeur quand la DEFinition est écrite. La valeur réelle ou variable est substituée à X quand la fonction est utilisée ; par exemple, elle peut être FNA (50).

2. Les Sous-programmes : un sous-programme est un groupe d'instructions à l'intérieur d'un programme BASIC qui a son nom propre et que l'on peut appeler autant de fois que l'on désire en écrivant simplement ce nom. Chaque fois que le nom du sous-programme est utilisé dans le corps du programme, toutes les instructions à l'intérieur du sous-programme sont exécutées. Les sous-programmes sont d'un emploi pratique pour exécuter à loisir une partie de programme sans avoir à répéter les instructions dans le détail. Voici un exemple de sous-programme :

```
500  REM SOUS-PROGRAMME MOYENNE
510  PRINT"JE VAIS CALCULER LA MOYENNE DE A ET DE B"
520  PRINT"A=";A;"B=";B
530  MOYENNE=(A+B)/2
540  PRINT"LA MOYENNE EST";MOYENNE
550  RETURN
```

Une fois que l'on a affecté une valeur à A et B, ce sous-programme peut être appelé par une instruction telle que :

```
50  GOSUB 500
```

Il peut ensuite être réutilisé dans le programme avec une instruction telle que :

```
170  GOSUB 500
```

et le résultat sera différent si A et B sont différents. Les sous-programmes clarifient le programme, le raccourcissent et vous font gagner du temps. On peut aussi mettre au point une bibliothèque de sous-programmes courants et les utiliser dans différents programmes. Il y a lieu, toutefois, d'être prudent en ce qui concerne les identificateurs de variables et numéros de ligne !

3. Les Opérateurs chaînes de caractères : ils permettent de manipuler du texte de façon pratique en opérant sur les chaînes de caractères. Les opérations sur les chaînes de caractères consistent à les modifier, les mesurer, les relier entre elles, les comparer, les décomposer, insérer du texte à gauche ou à droite, remplacer des caractères.
Ces opérateurs sont utiles dans les applications de traitement de texte.

4. Les Structures de données : le BASIC accepte les variables indicées avec un ou deux indices. Ces variables correspondent aux vecteurs mathématiques et aux matrices ou aux tableaux. L'utilisation de variables indicées permet d'établir des structures telles

que les listes et de se référer à n'importe quel élément de la liste. Par exemple, les éléments d'une liste intitulée CLIENT peuvent se référer à CLIENT(1), CLIENT(2), etc. et vous pouvez imprimer les dix valeurs en écrivant :

```
50  FOR N=1 TO 10
60  PRINT CLIENT(N)
70  NEXT N
```

Vous pouvez comparer deux éléments consécutifs en écrivant :

```
100  IF CLIENT(K) < CLIENT(K+1) THEN 500
```

5. Les Fichiers : Le BASIC de l'Amstrad offre des possibilités de stockage et/ou de récupération des données à partir de fichiers de disque. Ces possibilités sont décrites dans la documentation fournie par le constructeur.

6. Les Ressources supplémentaires : Le BASIC de l'Amstrad offre des ressources supplémentaires telles que les instructions ON ... GOTO (SELON ... ALLER), READ ... DATA (LIRE ... DONNEES). Etudiez-les dans le manuel ou dans un livre car elles ne peuvent que faciliter la programmation.

Certaines abréviations sont autorisées. Vous pouvez taper par exemple :

```
? 25 * 32
```

au lieu de

```
PRINT 25 * 32
```

Il s'agit là d'une fonction pratique vous permettant d'utiliser votre ordinateur comme une calculatrice de poche.

De plus, vous disposez d'un mode double précision pour améliorer la précision des calculs numériques, et de possibilités de tabulation (TAB) pour faciliter la présentation des tables.

Enfin, rappelons l'existence de commandes particulières pour

générer des graphiques, produire des effets sonores et faciliter les modifications à l'intérieur d'un programme. Il s'agit en particulier de commande de l'écran et d'établissement de points d'arrêt pour que le programme stoppe automatiquement.

Conclusion

L'objectif de ce livre a été de vous apprendre avec rapidité et efficacité la façon d'écrire vos premiers programmes en BASIC sur Amstrad. Si le BASIC vous intéresse vraiment, vous souhaiterez en savoir davantage. Nous vous conseillons de faire tous les exercices et de créer vos propres programmes.

Au fur et à mesure que vous progresserez, vous aurez envie d'apprendre des techniques plus élaborées. Vous trouverez, à la fin de ce livre, une liste d'ouvrages sur la programmation.

Reconnaissez que l'apprentissage du BASIC est facile et agréable. Nous souhaitons que vous ayiez envie d'aller plus loin. Si vous jugez que cet ouvrage est susceptible d'être amélioré, nous recevrons avec plaisir vos commentaires et suggestions.



A N N E X E A

Réponses aux exercices

2

2.1 : 10 PRINT"BONNE JOURNEE"

2.2 :
```
10 PRINT "AAAAA"
20 PRINT "BBBB"
30 PRINT "CCC"
40 PRINT "DD"
50 PRINT "E"
```

2.3 :
```
20 PRINT"*******"
30 PRINT"*TITRE*"
40 PRINT"*******"
```

2.4 : **a)** En BASIC, une étiquette est un numéro de ligne, et doit précéder chaque instruction d'un programme.

b) Le mode d'exécution différé est le mode normal d'entrée d'un programme. Les instructions BASIC sont tapées avec des numéros de ligne et mémorisées par l'ordinateur pour être exécutées ultérieurement.

c) L'exécution immédiate est le mode qui permet de taper une instruction sans numéro de ligne et de l'exécuter immédiatement. On l'appelle aussi mode calculateur.

d) Une instruction vide est une instruction qui ne fait rien. Elle est représentée par un numéro de ligne (ou étiquette) qui apparaît seul - sans rien d'autre sur la ligne - sauf une éventuelle instruction «REM».

e) Un curseur est un symbole visuel spécial sur l'écran (un carré mobile) qui indique la position courante. Il clignote pour renforcer la lisibilité.

f) Un mot réservé est un nom qui a une signification bien précise en BASIC. Il ne peut être utilisé comme variable par le programmeur.

g) L'indicatif est un caractère ou un message généré par un programme qui indique que le programme attend que l'utilisateur entre l'infor-

mation. Le BASIC de l'Amstrad utilise l'indicatif Ready qui signifie «tapez votre instruction suivante». L'indicatif «?» signifie «donnez-moi une entrée».

2.5 : PRINT est l'instruction qui permet d'afficher les caractères sur l'écran.

2.6 : Oui, mais c'est une façon plutôt fastidieuse de le faire puisqu'il faut retaper les instructions si vous voulez relancer le programme. De même, les sauts et les branchements conditionnels (que nous étudierons plus tard) ne peuvent fonctionner de cette façon.

2.7 : Afin d'effacer le contenu de la mémoire (RAM).

2.8 : Oui. Le BASIC va insérer chaque instruction dans le bon ordre dans le programme de sorte que les étiquettes seront classées par ordre numérique croissant.

2.9 : Non ; vous devez taper : PRINT "EXEMPLE".

2.10 : La touche ENTER permet de valider une commande.

2.11 : Pour effacer l'instruction 20 dans un programme, tapez une instruction vide portant le numéro 20.

2.12 : Non ; la nouvelle ligne se substituera à l'ancienne.

2.13 :

```
10 PRINT"OOOOO    U   U    I"
20 PRINT"O   O    U   U    I"
30 PRINT"O   O    U   U    I"
40 PRINT"O   O    U   U    I"
50 PRINT"O   O    U   U    I"
60 PRINT"O   O    U   U    I"
70 PRINT"OOOOO    UUUUU    I"
```

3

3.1 : PRINT((5+6)/(1+2))/3

3.2 : PRINT 1+(1/2) * (1/(1+(1/2)))

ou

PRINT 1 + .5/(1+.5)

3.3 : PRINT 100 / 1.6

3.4 : PRINT 350/95

4

4.1 :

```
10   INPUT A,B,C,D
20   SOMME=A+B+C+D
30   MOYENNE=SOMME/4
40   PRODUIT=A*B*C*D
50   PRINT SOMME,MOYENNE,PRODUIT
60   END
```

4.2 :

a) non	e) oui	i) oui
b) oui	f) oui	j) non
c) non	g) non	k) oui
d) oui	h) non	l) oui

4.3 :

```
10   PRINT"DONNEZ VOTRE PRENOM"
20   INPUT PRENOM$
30   PRINT"JE PENSE CONNAITRE UN ";PRENOM$
```

4.4 :

```
10   PRINT"DONNEZ-MOI LE NOM D'UN OBJET";
20   INPUT O$
30   PRINT"LE NOM D'UN MEUBLE";
40   INPUT M$
50   PRINT"LE NOM D'UN AMI";
60   INPUT A$
70   PRINT
80   PRINT"EST-CE QUE VOTRE AMI";A$;"A UN(E)";O$;
```

```
        "SUR UN(E)"; M$"?"
90  END
```

4.5 : b et c sont corrects

5

5.1 : Utilisation de blancs. Utilisation de la touche CLR/HOME. Espacement des opérations.

5.2 : a) oui   d) oui
b) oui   e) non
c) oui   f) non

5.3 : Afin de donner des indications à l'utilisateur.

5.4 :
```
INPUT"VOTRE NOM:";NOM$
INPUT"2+3=";C
INPUT"LES 2 NOMBRES SONT:";A,B
```

5.5 : Pour améliorer la compréhension du programme.

5.6 : A = 3

6

6.1 : L'instruction IF permet au programme de prendre des décisions, en changeant ainsi son déroulement selon les variations de l'entrée des données ou des valeurs calculées.

6.2 : Entrée de la réponse. Test oui ou non (20-30). Affichage réponses possibles (40).

6.3 : a) oui   d) oui
b) oui   e) oui
c) oui   f) non
g) non

6.4 : oui

6.5 : Une boucle de programme exécute de façon répétitive une partie de programme. Pour éviter d'avoir une boucle infinie (une boucle qui répète

indéfiniment) il faut y inclure un test qui, quand il réussit, permet au programme de sortir de la boucle.

7.1 :

```
10  PRINT"AFFICHAGE DES DIX PREMIERS NOMBRES"
20  FOR I=1 TO 10
30  PRINT I
40  NEXT I
```

7.2 :

```
10  INPUT"HEURES,MINUTES:";HEURES,MINUTES
20  REM VALIDER INPUT
30  IF HEURES> =0 AND HEURES<72 AND MINUTES> =
    0 AND MINUTES<60 THEN 70
40  PRINT"INCORRECT,NOUVEL ESSAI"
50  GOTO 10
60  REM AFFICHAGE LIGNE D'HEURES
70  IF HEURES=0 THEN 110
80  FOR A=1 TO HEURES
90  PRINT "H";
100 NEXT A
110 PRINT
120 REM AFFICHAGE LIGNE MINUTES
130 IF MINUTES=0 THEN 170
140 FOR A=1 TO MINUTES
150 PRINT"M";
160 NEXT A
170 PRINT
180 END
```

7.3 : On peut effectuer un saut vers l'intérieur d'une boucle si celle-ci n'est pas définie par des instructions FOR...NEXT.

7.4 :

```
10  PRINT"PROGRAMME SOMME DES N PREMIERS NOMBRES"
20  PRINT"IMPAIRS"
30  INPUT"NOMBRE A ADDITIONNER";NOMBRE
40  REM TEST DE VALIDATION DE L'ENTREE
```

```
50  IF NOMBRE>0 AND NOMBRE<1000 THEN 100
60  PRINT"NOMBRE INCORRECT..NOUVEL ESSAI"
70  GOTO 30
80  REM FAIRE TABLE
100 PRINT"NOMBRE","SOMME"
110 PRINT
120 FOR N=1 TO NOMBRE
130 SOMME=SOMME+2*N-1
140 PRINT N,SOMME
150 NEXT N
160 END
```

7.5 :

```
10  INPUT "TAUX DE LA TAXE %";TAXE
20  IF TAXE<1 OR TAXE>100 THEN 10
30  PRINT"PRIX","TAXE","PRIX+TAXE"
40  FOR PRIX =10 TO 1000 STEP 10
50  PRINT PRIX;PRIX*TAXE/100,PRIX+PRIX*TAXE/100
60  NEXT PRIX
70  END
```

8

8.1 : Le codage est une phase de la programmation. La programmation englobe la définition de l'algorithme, le codage, la mise au point et la documentation.

8.2 : Une variable est pistée en insérant des instructions PRINT pour montrer la valeur de la variable en des points critiques dans le programme.

8.3 : Un organigramme est un diagramme symbolique qui montre la suite des événements qui ont lieu durant l'exécution du programme.

8.4 : Des programmes écrits de façon claire sont faciles à comprendre, et ainsi faciles à modifier, ce qui permet au programmeur qui a créé le programme ainsi qu'aux autres programmeurs, de le modifier facilement.

ANNEXE B

Les mots réservés courants du BASIC

Cette liste vous permettra de ne pas utiliser des noms de variables illégaux.

ABS
AFTER
ASC
ATN
AUTO
CALL
CAT
CHAIN
CHAIN MERGE
CHR$
CINT
CLEAR
CLG
GLOSEIN
CLOSEOUT
CLS
CONT
COS
CREAL
DATA
DEC$
DEF FN
DEFINT
DEFREAL
DEFSTR
DEG
DELETE
DI
DIM
DRAW
DRAWR
EDIT
EL
END
ENT
ENV
EOF
ERASE
ERL
ERR
ERROR
EVERY
EXP
FIX
FOR
FRE
GOSUB
GOTO
HEX$
HIMEM
IF
INK
INKEY
INKEY$
INP
INPUT
INSTR
INT
JOY
KEY DEF
KEY
LEFT$
LEN
LET
LINE INPUT
LIST
LOAD
LOCATE
LOG
LOG1()
LOWER$
MAX
MEMORY
MERGE
MID$
MIN
MODE
MOVE
MOVER
NEW
NEXT
ON BREAK GOSUB
ON BREAK STOP
ON ERROR GOTO
ON SQ GOSUB
ON(exp)GOSUB
ON(exp)GOTO
OPENIN
OPENOUT
ORIGIN
OUT
PAPER
PEEK
PEN
PI
PLOT
PLOTR
POKE
POS
PRINT
RAD
RANDOMIZE
READ
RELEASE
REM
REMAIN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RND
ROUND
RUN
SAVE
SGN
SIN
SOUND
SPACE$
SPEED INK
SPEED KEY
SPEED WRITE
SQ
SQR
STOP
STR$
STRING$
SYMBOL AFTER
SYMBOL
TAG
TAGOFF
TAN
TEST
TESTR
TIME
TROFF
TRON
UNT
UPPER$
VAL
VPOS
WAIT
WEND
WHILE
WIDTH
WINDOW SWAP
WINDOW
WRITE
XPOS
YPOS
ZONE

ANNEXE C

Glossaire BASIC

Affectation Opération qui consiste à donner une valeur à une variable, indiquée par le symbole «=» en BASIC.

Algorithme Séquence d'étapes qui spécifie la solution à un problème donné.

Alphanumérique Ensemble des caractères alphabétiques et numériques.

BASIC *Beginners All-purpose Symbolic Instruction Code* (Langage de programmation «pour débutants» et «pour tous usages») Langage de programmation évolué conçu pour faciliter l'apprentissage.

Binaire Système de numérotation utilisant seulement 2 chiffres : 0 et 1.

Bit Contraction de *Binary Digit,* chiffre binaire. Il peut prendre la valeur 0 ou 1.

Boucle Séquence d'instructions d'un programme exécutée de façon répétitive jusqu'à ce qu'un événement spécifique ait lieu, en général jusqu'à ce qu'une variable atteigne une valeur donnée.

Boucle imbriquée Une boucle située à l'intérieur d'une autre boucle.

Boucle infinie Une boucle qui n'a pas de point de sortie et s'exécute indéfiniment. C'est une erreur courante des programmeurs lorsqu'aucun test conditionnel n'est inclus dans la boucle ou lorsque le test réussit (ou échoue) toujours. Sortir d'une boucle infinie nécessite l'utilisation d'une commande de «sortie» spéciale - CTRL-C en général.

Bug Erreur. Mieux vaut les prévenir que les guérir.

Chaîne Suite de caractères (par opposition à «nombre»). En BASIC, le nom d'une variable est différent selon qu'il s'agit d'une chaîne ou d'un nombre. Par exemple, MOT$ est une variable chaîne de caractères alors que NOMBRE est une variable numérique.

Chargement Transfert des données ou d'un programme dans la mémoire interne de l'ordinateur.

Circuit intégré Appelé aussi CI. Circuit électronique comprenant de nombreux transistors et fonctions logiques sur une petite plaque de silicium.

Codage Opération qui consiste à traduire un algorithme ou un organigramme en une série d'instructions d'un programme. C'est une des étapes du processus de programmation.

Commande Mot réservé utilisé pour exécuter une fonction de base telle qu'effacer l'écran, démarrer un pro-

gramme ou accéder aux fichiers. En spécifiant une commande, elle active un programme spécialisé dans l'ordinateur pour effectuer cette tâche.

CPU *(Central Processing Unit)* Voir Unité Centrale de Traitement.

CRT *(Cathode Ray Tube)* Tube à Rayons Cathodiques.

Curseur Symbole utilisé pour indiquer la position courante où le caractère doit être affiché sur l'écran. C'est le plus souvent une marque mobile ou un tiret. Il existe en général des touches spéciales pour positionner correctement le curseur sur l'écran.

Disque Support magnétique sur lequel sont stockés les données et les programmes. L'information est stockée sur disque sous forme de fichiers qui peuvent être retrouvés en les appelant par leur nom. Les disques peuvent contenir une grande quantité d'information. Ils forment un dispositif de mémoire de masse utilisé pour les petits ordinateurs.

Disquette Disque souple, c'est-à-dire un disque de 8 pouces ou de 5 pouces 1/4 conçu pour stocker les programmes et les données de façon économique.

Données Tout texte ou tout nombre qu'un programme peut traiter.

Double précision Nombre comportant deux fois plus de chiffres que dans la représentation normale. Chaque interpréteur représente les nombres avec un ensemble de chiffres. La double précision sert à faire des calculs scientifiques ou des calculs financiers qui nécessitent des nombres importants ou des calculs intensifs.

Editeur Programme conçu pour faciliter l'entrée et la modification d'un texte. Il est utilisé pour corriger les erreurs faites en tapant un programme.

E/S Entrée/Sortie. Les communications avec l'ordinateur et de l'ordinateur vers le monde extérieur.

Etiquette Numéro de ligne en BASIC.

Expression Combinaison d'opérandes ou de variables séparés par des opérateurs. Une expression représente une formule et exécute un calcul spécifique. Quand une expression est évaluée par l'interpréteur lors de l'exécution, le résultat est une valeur.

Expression logique Combinaison d'opérandes ou de variables séparés par des opérateurs relationnels (=, > etc.). La valeur d'une expression est soit vraie, soit fausse.

Fichier Collection d'informations à laquelle on a attribué un nom. Un programme est en général stocké sur un disque en tant que fichier.

Graphique Image, figure ou dessin affiché sur l'écran utilisant en général un schéma de petits points adjacents. L'utilisation de la couleur améliore l'aspect des graphiques.

Hardware Voir Matériel.

IC *(Integrated Circuit)* Voir Circuit intégré.

Initialisation La phase d'un programme durant laquelle des valeurs initiales sont affectées aux variables. Chaque boucle nécessite une phase d'initialisation.

Instruction Ordre donné à l'interpréteur qui va affecter les données sur lesquelles on opère. Chaque instruction précédée d'un numéro de ligne est une partie de programme (les commandes n'affectent pas les valeurs des données). Elles exécutent les fonctions de base ou facilitent l'élaboration ou l'utilisation d'un programme.

Instruction vide Instruction qui ne fait rien. Par exemple, l'instruction : 10 REM peut être utilisée pour générer un espace dans le programme et en améliorer la lisibilité.

Interface Circuits électroniques qui permettent de connecter un dispositif spécifique à l'ordinateur. Un disque, une imprimante ou un magnétophone nécessitent des interfaces spécifiques.

Interpréteur Programme chargé de traduire les instructions d'un langage de programmation (BASIC par exemple) en langage binaire de l'ordinateur et de les exécuter. Une fois que l'interpréteur est dans l'ordinateur, l'ordinateur comprend les instructions.

K 1 K égale 1024 octets. (Lire K ou Kilo.) K est utilisé pour indiquer la taille de la mémoire, en général en octets.

Langage évolué Langage de programmation conçu pour donner des ordres à l'ordinateur. BASIC est un langage évolué.

Langage machine Langage binaire que l'ordinateur comprend directement, c'est-à-dire un ensemble d'instructions limité qui manipule des informations binaires dans l'unité centrale et la mémoire.

Logiciel Les programmes.

Logique Variable ou expression qui prend la valeur vrai ou faux.

Matériel L'équipement y compris l'ordinateur, les disques et tout autre support qui composent le système informatique par opposition au logiciel (instructions ou programmes).

Mémoire Support contenant l'information. La mémoire interne réside en général sur la même carte que l'unité centrale et mémorise les programmes et les données sous forme d'octets. Les unités de mémoire de masse sont les disquettes et les cassettes.

Mémoire morte Mémoire qui ne peut être modifiée par le programmeur. Elle contient en général une partie ou l'intégralité du moniteur et parfois un interpréteur BASIC simple.

Mémoire vive Partie modifiable de la mémoire de l'ordinateur (le reste étant la mémoire morte).

Micro-ordinateur Ordinateur qui utilise un microprocesseur comme unité centrale de traitement.

Microprocesseur Circuit intégré qui exécute la plupart des fonctions d'une unité centrale sur une seule puce. Aujourd'hui un microproces-

seur peut contenir dans certains cas des dizaines de milliers de transistors sur un seul circuit et peut même contenir la mémoire.

Mise au point Opération qui consiste à enlever les erreurs d'un programme. Cette opération peut s'avérer ardue ; tous les efforts doivent donc porter sur l'élaboration du programme pour qu'il y ait le moins d'erreurs possible.

Moniteur Programme minimum servant à faire fonctionner un système informatique. Le moniteur lit les caractères à partir du clavier, les affiche à l'écran, et exécute les transferts de données entre le clavier, l'écran et les périphériques.

Mot réservé Mot qui a une signification prédéterminée pour l'interpréteur BASIC. Il ne peut être utilisé comme nom de variable par le programmeur.

MPU Unité de microprocesseur.

Nombre à virgule fixe Nombre où la virgule décimale a une position fixe.

Nombre à virgule flottante Un nombre décimal. Un nombre fixe de chiffres est utilisé pour le codage interne de tout nombre ; lors des calculs sur ce nombre, la virgule «flotte» de droite à gauche selon les résultats.

Non résident Programme qui ne se trouve pas normalement dans la mémoire permanente (ROM) de l'ordinateur. Un programme non résident peut être stocké sur une cassette ou une disquette.

Octet Groupe de 8 bits.

Opérateur Symbole représentant toute opération correcte sur des valeurs (+ (addition), * (multiplication), etc.).

Opérateur relationnel Opérateur logique qui établit une relation de grandeur entre des valeurs.

Ordinateur Ensemble contenant au moins une unité centrale, une mémoire, les interfaces de base qui lui permettent de communiquer avec le monde extérieur et une alimentation. Il peut aussi comporter un clavier, un écran et des unités de disques. Un ordinateur peut stocker des programmes et les exécuter. Il communique avec le monde extérieur au moyen de dispositifs d'entrées/sorties. Le dispositif d'entrée courant est le clavier et le dispositif de sortie est l'écran ou bien l'imprimante. La mémoire supplémentaire est fournie par les unités de disques ou des enregistreurs de cassettes.

Organigramme Représentation visuelle symbolique d'un algorithme.

Périphérique Dispositif relié à l'ordinateur tel qu'une imprimante, une unité de disque ou un terminal.

Programme Suite ordonnée d'instructions établie en vue d'être exécutée par l'ordinateur. Chaque programme est écrit dans un langage de programmation défini et doit être chargé dans la mémoire de l'ordinateur afin d'être exécuté.

Puce (*Chip* composant). Circuit intégré qui réside sur une plaque de sili-

cium montée dans un boîtier en plastique ou en céramique.

RAM Voir Mémoire vive.

Résident Programme stocké en permanence dans la mémoire de l'ordinateur (la ROM).

ROM Voir Mémoire morte.

RUN Commande qui permet de démarrer l'exécution d'un programme.

Saut Branchement, c'est-à-dire obligation d'exécuter une instruction hors de la séquence.

Syntaxe Ensemble des règles qui définissent les instructions acceptables du langage d'un ordinateur. La syntaxe du BASIC est simple, l'interpréteur recherche et indique les erreurs de syntaxe.

Système d'exploitation Programme auxiliaire disposant de larges possibilités pour exécuter tous les transferts de données courants et le traitement des données exigé pour utiliser de façon efficace les ressources de l'ordinateur. Le système d'exploitation gère les fichiers disque, exécute les conversions de format et fait démarrer et arrêter les programmes.

Terminal Combinaison d'un écran et d'un clavier ou d'une imprimante et d'un clavier, utilisée pour communiquer de façon efficace avec un ordinateur.

Unité Centrale de Traitement Module électronique chargé de prendre les instructions stockées dans la mémoire, de les décoder, de les exécuter dans l'ordre approprié. Pour la plupart des petits ordinateurs, l'unité centrale et la mémoire résident sur une seule plaquette de circuit imprimé ou «carte». L'unité centrale utilise un microprocesseur et quelques autres composants.

Unité de disque Mécanisme utilisé pour lire et écrire sur un disque.

Variable Emplacement mémoire qui a un nom et qui peut prendre des valeurs successives dans le temps. Le BASIC fait la distinction entre variable chaîne de caractères et variable numérique. Le nom d'une variable chaîne de caractères doit se terminer par un «$» (dollar).

INDEX

LA BIBLIOTHÈQUE SYBEX

OUVRAGES GÉNÉRAUX

VOTRE PREMIER ORDINATEUR *par Rodnay Zaks,*
296 pages, Réf. 394

VOTRE ORDINATEUR ET VOUS *par Rodnay Zaks,*
296 pages, Réf. 271

DU COMPOSANT AU SYSTÈME : une introduction aux microprocesseurs *par Rodnay Zaks,*
636 pages, Réf. 340

TECHNIQUES D'INTERFACE aux microprocesseurs *par Austin Lesea et Rodnay Zaks,*
450 pages, Réf. 339, 3ème édition

LEXIQUE INTERNATIONAL MICROORDINATEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. 234

GUIDE DES MICRO-ORDINATEURS A MOINS 3 000 F *par Joël Poncet,*
144 pages, Réf. 322

LEXIQUE MICRO-INFORMATIQUE *par Pierre Le Beux,*
140 pages, Réf. 369

BASIC

VOTRE PREMIER PROGRAMME BASIC *par Rodnay Zaks,*
208 pages, Réf. 263

INTRODUCTION AU BASIC *par Pierre Le Beux,*
336 pages, Réf. 335

LE BASIC PAR LA PRATIQUE : 60 exercices *par Jean-Pierre Lamoitier,*
252 pages, Réf. 395

LE BASIC POUR L'ENTREPRISE *par Xuan Tung Bui,*
204 pages, Réf. 253, 2ème édition

PROGRAMMES EN BASIC, Mathématiques, Statistiques, Informatique *par Alan R. Miller,*
318 pages, Réf. 259

AU COEUR DES JEUX EN BASIC *par Richard Mateosian,*
352 pages, Réf. 233

JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
192 pages, Réf. 246

NOUVEAUX JEUX D'ORDINATEUR EN BASIC *par David H. Ahl,*
204 pages, Réf. 247

PASCAL

INTRODUCTION AU PASCAL *par Pierre Le Beux,*
496 pages, Réf. 330

LE PASCAL PAR LA PRATIQUE *par Pierre Le Beux et Henri Tavernier,*
562 pages, Réf. 361

LE GUIDE DU PASCAL *par Jacques Tiberghien,*
504 pages, Réf. 232

PROGRAMMES EN PASCAL pour Scientifiques et Ingénieurs *par Alan R. Miller,*
392 pages, Réf. 240

AUTRES LANGAGES

INTRODUCTION A ADA *par Pierre Le Beux,*
366 pages, Réf. 360

MICROORDINATEUR

ALICE

JEUX EN BASIC POUR ALICE *par Pierre Monsaut,*
96 pages, Réf. 320

APPLE / MACINTOSH

PROGRAMMEZ EN BASIC SUR APPLE II *par Léopold Laurent,*
208 pages, Réf. 333

APPLE II 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 283

JEUX EN PASCAL SUR APPLE *par Douglas Hergert et Joseph T. Kalash,*
372 pages, Réf. 241

GUIDE DU BASIC APPLE II *par Douglas Hergert,*
272 pages, Réf. 306

APPLE II, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 373

MACINTOSH, GUIDE DE L'UTILISATEUR *par Joseph Caggiano,*
208 pages, Réf. 396

ATARI

JEUX EN BASIC SUR ATARI *par Paul bunn,*
96 pages, Réf. 282

ATMOS

JEUX EN BASIC SUR ATMOS *par Pierre Monsaut,*
96 pages, Réf. 346

ATMOS, 56 PROGRAMMES *par Stanley R. Trost,*
180 pages, Réf. 372

COMMODORE 64

JEUX EN BASIC SUR COMMODORE 64 *par Pierre Monsaut,*
96 pages, Réf. 317

COMMODORE 64, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 342

GUIDE DU BASIC VIC 20, COMMODORE 64 *par Douglas Hergert,*
240 pages, Réf. 312

COMMODORE 64, GUIDE DE L'UTILISATEUR *par J. Kassmer,*
144 pages, Réf. 314

COMMODORE 64, 66 PROGRAMMES *par Stanley R. Trost,*
192 pages, Réf. 319

DRAGON

JEUX EN BASIC SUR DRAGON *par Pierre Monsaut,*
96 pages, Réf. 324

GOUPIL

PROGRAMMEZ VOS JEUX SUR GOUPIL *par François Abella,*
208 pages, Réf. 264

IBM

IBM PC EXERCICES EN BASIC *par Jean-Pierre Lamoitier,*
256 pages, Réf. 338

IBM PC Guide de l'utilisateur *par Joan Lasselle et Carol Ramsey,*
160 pages, Réf. 301

IBM PC 66 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 280

LASER

LASER JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 371

MO5

MO5 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 367

MO5, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 370

ORIC

JEUX EN BASIC SUR ORIC *par Peter Shaw,*
96 pages, Réf. 278

ORIC PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 344

SHARP

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2 *par Michel Lhoir,*
2 tomes, Réf. 261-262

SPECTRUM

PROGRAMMEZ EN BASIC SUR SPECTRUM *par S.M. Gee,*
208 pages, Réf. 252

JEUX EN BASIC SUR SPECTRUM *par Peter Shaw,*
96 pages, Réf. 276

SPECTRUM, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 381

SPECTRUM JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 368

TI 99/4

PROGRAMMEZ VOS JEUX SUR TI 99/4 *par François Abella,*
160 pages, Réf. 303

TO 7

JEUX EN BASIC SUR TO 7 *par Pierre Monsaut,*
96 pages, Réf. 326

TO 7, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 328

TIMEX

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 256

TRS-80

PROGRAMMEZ EN BASIC SUR TRS-80 *par Léopold Laurent,*
2 tomes, Réf. 250-251

DÉCOUVREZ LE SHARP PC-1500 ET LE TRS-80 PC-2 *par Michel Lhoir,*
2 tomes, Réf. 366-262

JEUX EN BASIC SUR TRS-80 MC-10 *par Pierre Monsaut,*
96 pages, Réf. 323

JEUX EN BASIC SUR TRS-80 *par Chris Palmer,*
96 pages, Réf. 302

JEUX EN BASIC SUR TRS-80 COULEUR *par Pierre Monsaut,*
96 pages, Réf. 325

TRS-80 MODÈLE 100, GUIDE DE L'UTILISATEUR *par Orson Kellog,*
112 pages, Réf. 300

VIC 20

PROGRAMMEZ EN BASIC SUR VIC 20 *par G. O. Hamann,*
2 tomes, Réf. 244-337

JEUX EN BASIC SUR VIC 20 *par Alastair Gourlay,*
96 pages, Réf. 277

VIC 20, PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 341

GUIDE DU BASIC VIC 20, COMMODORE 64 *par Douglas Hergert,*
240 pages, Réf. 312

VIC 20 JEUX D'ACTION *par Pierre Monsaut,*
96 pages, Réf. 345

ZX 81

ZX 81 GUIDE DE L'UTILISATEUR *par Douglas Hergert,*
208 pages, Réf. 256

ZX 81 56 PROGRAMMES BASIC *par Stanley R. Trost,*
192 pages, Réf. 304

GUIDE DU BASIC ZX 81 *par Douglas Hergert,*
204 pages, Réf. 285

JEUX EN BASIC SUR ZX 81 *par Mark Charlton,*
96 pages, Réf. 275

ZX 81 PREMIERS PROGRAMMES *par Rodnay Zaks,*
248 pages, Réf. 343

MICRO-PROCESSEURS

PROGRAMMATION DU Z80 *par Rodnay Zaks,*
618 pages, Réf. 220

APPLICATIONS DU Z80 *par James W. Coffron,*
304 pages, Réf. 274

PROGRAMMATION DU 6502 *par Rodnay Zaks,*
376 pages, Réf. 331, 2ème édition

APPLICATIONS DU 6502 *par Rodnay Zaks,*
288 pages, Réf. 332

PROGRAMMATION DU 6800 *par Daniel-Jean David et Rodnay Zaks,*
374 pages, Réf. 327

PROGRAMMATION DU 6809 *par Rodnay Zaks et William Labiak,*
392 pages, Réf. 328

PROGRAMMATION DU 8086/8088 *par James W. Coffron,*
304 pages, Réf. 316

MISE EN OEUVRE DU 68000 *par C. Vieillefond,*
352 pages, Réf. 363

SYSTÈMES D'EXPLOITATION

GUIDE DU CP/M AVEC MP/M *par Rodnay Zaks,*
354 pages, Réf. 336

CP/M APPROFONDI *par Alan R. Miller,*
380 pages, Réf. 334

INTRODUCTION AU p-SYSTEM UCSD *par Charles W. Grant et Jon Butah,*
308 pages, Réf. 365

GUIDE DU PC DOS *par Richard A. King,*
240 pages, Réf. 313

LOGICIELS ET APPLICATIONS

INTRODUCTION AU TRAITEMENT DE TEXTE *par Hal Glatzer,*
228 pages, Réf. 243

INTRODUCTION A WORDSTAR *par Arthur Naiman,*
200 pages, Réf. 255

WORDSTAR APPLICATIONS *par Julie Anne Arca,*
320 pages, Réf. 305

VISICALC APPLICATIONS *par Stanley R. Trost,*
304 pages, Réf. 258

VISICALC POUR L'ENTREPRISE *par Dominique Helle,*
304 pages, Réf. 309

INTRODUCTION A dBASE II *par Alan Simpson,*
280 pages, Réf. 364

DE VISICALC A VISI ON *par Jacques Bourdeu,*
256 pages, Réf. 321

La plupart de ces ouvrages existent en version anglaise. N'hésitez pas à demander notre catalogue.

EN ANGLAIS

BASIC EXERCISES FOR APPLE *by JEAN-PIERRE LAMOITIER,*
232 pages, Réf. 0-084

BASIC FOR BUSINESS *by DOUGLAS HERGERT,*
224 pages, Réf. 0-080

CELESTIAL BASIC : Astronomy on your Computer *by ERIC BURGESS,*
228 pages, Réf. 0-087

INTRODUCTION TO PASCAL (Including UCSD Pascal) *by RODNAY ZAKS,*
422 pages, Réf. 0-066

DOING BUSINESS WITH PASCAL *by RICHARD HERGERT AND DOUGLAS HERGERT,*
380 pages, Réf. 0-091

MASTERING VISICALC *by DOUGLAS HERGERT,*
224 pages, Réf. 0-090

THE APPLE CONNECTION *by JAMES W. COFFRON,*
228 pages, Réf. 0-085

PROGRAMMING THE Z8000 *by RICHARD MATEOSIAN,*
300 pages, Réf. 0-032

A MICROPROGRAMMED APL IMPLEMENTATION *by RODNAY ZAKS,*
350 pages, Réf. 0-005

ADVANCED 6502 PROGRAMMING *by RODNAY ZAKS,*
292 pages, Réf. 0-089

FORTRAN PROGRAMS FOR SCIENTISTS AND ENGINEERS *by ALAN R. MILLER,*
320 pages, Réf. 0-082

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 203.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Volgelsanger. WEG 111
4000 Düsseldorf 30
Post Bos N° 30.09.61
Tel. : (0211) 626441
Telex : 08588163

ANGLETERRE
Unit 4 - Bourne Industrial Park
Bourne Road, Crayford
Kent DA1 4BU
Tel. : (0322) 57717



Paris • Berkeley • Düsseldorf • Londres

Achevé d'imprimer le 22 octobre 1984 sur les presses de l'Imprimerie «La Source d'Or»
63200 Marsat - Dépôt légal : 4e trimestre 1984 - Imprimeur n° 1622



PREMIERS PROGRAMMES

Ecrivez votre premier programme BASIC sur Amstrad en moins d'une heure !
D'une présentation claire, comportant de nombreux diagrammes et illustrations en couleur, ce livre vous enseigne les bases de la programmation en BASIC sur Amstrad. Avec lui, vous apprendrez à programmer en quelques heures, quels que soient votre âge et votre formation. Aucune expérience préalable de la programmation n'est nécessaire.



Rodnay Zaks, docteur en informatique de l'université de Californie, Berkeley, a été un des pionniers de l'enseignement de la micro-informatique et a donné sur ce sujet de nombreuses conférences à travers le monde. Il est l'auteur d'un grand nombre de best-sellers sur la programmation et les microprocesseurs, maintenant disponibles dans plus de dix langues.

ISBN 2-7361-0105-9 405 1184 98 F


SYBEX
RODNAY ZAKS
AMSTRAD
PREMIERS PROGRAMMES



Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE



https://acpc.me/